# Mastering CodeWarrior for Windows 95/NT

*The Official Guide*

# Mastering™ CodeWarrior® for Windows 95/NT®
## *The Official Guide*

**Jim Trudeau**

**San Francisco • Paris • Düsseldorf • Soest**

Associate Publisher: Gary Masters
Acquisitions Manager: Kristine Plachy
Acquisitions & Developmental Editor: Suzanne Rotondo
Editor: Michael D. Welch
Project Editors: Maureen Adams and LeeAnn Pickrell
Technical Editor: Lucien Stavenhagen
Book Designer: Catalin Dulfu
Electronic Publishing Specialist: Kate Kaminski
Production Coordinator: Amy Eoff
Indexer: Ted Laux
Cover Designer: Design Site
Cover Illustrator: Metrowerks

Screen reproductions produced with Collage Complete.
Collage Complete is a trademark of Inner Media Inc.

SYBEX is a registered trademark of SYBEX Inc.
Mastering is a trademark of SYBEX Inc.

Internet Explorer copyright Microsoft Corporation, 1996.
All rights reserved.

TRADEMARKS: SYBEX has attempted throughout this book to distinguish proprietary trademarks from descriptive terms by following the capitalization style used by the manufacturer.

The author and publisher have made their best efforts to prepare this book, and the content is based upon final release software whenever possible. Portions of the manuscript may be based upon pre-release versions supplied by software manufacturer(s). The author and the publisher make no representation or warranties of any kind with regard to the completeness or accuracy of the contents herein and accept no liability of any kind including but not limited to performance, merchantability, fitness for any particular purpose, or any losses or damages of any kind caused or alleged to be caused directly or indirectly from this book.

Copyright ©1997 SYBEX Inc., 1151 Marina Village Parkway, Alameda, CA 94501. World rights reserved. No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission of the publisher.

Library of Congress Card Number: 96-72281
ISBN: 0-7821-2057-1
Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

# Software License Agreement: Terms and Conditions

The media and/or any online materials accompanying this book that are available now or in the future contain programs and/or text files (the "Software") to be used in connection with the book. SYBEX hereby grants to you a license to use the Software, subject to the terms that follow. Your purchase, acceptance, or use of the Software will constitute your acceptance of such terms.

The Software compilation is the property of SYBEX unless otherwise indicated and is protected by copyright to SYBEX or other copyright owner(s) as indicated in the media files (the "Owner(s)"). You are hereby granted a single-user license to use the Software for your personal, noncommercial use only. You may not reproduce, sell, distribute, publish, circulate, or commercially exploit the Software, or any portion thereof, without the written consent of SYBEX and the specific copyright owner(s) of any component software included on this media.

In the event that the Software or components include specific license requirements or end-user agreements, statements of condition, disclaimers, limitations or warranties ("End-User License"), those End-User Licenses supersede the terms and conditions herein as to that particular Software component. Your purchase, acceptance, or use of the Software will constitute your acceptance of such End-User Licenses. Please refer to the Metrowerks End-User Software License in the back of this book for specific requirements of the CodeWarrior Lite software.

By purchase, use or acceptance of the Software you further agree to comply with all export laws and regulations of the United States as such laws and regulations may exist from time to time.

## Software Support

Components of the supplemental Software and any offers associated with them may be supported by the specific Owner(s) of that material but they are not supported by SYBEX. Information regarding any available support may be obtained from the Owner(s) using the information provided in the appropriate read.me files or listed elsewhere on the media.

Should the manufacturer(s) or other Owner(s) cease to offer support or decline to honor any offer, SYBEX bears no responsibility. This notice concerning support for the Software is provided for your information only. SYBEX is not the agent or principal of the Owner(s), and SYBEX is in no way responsible for providing any support for the Software, nor is it liable or responsible for any support provided, or not provided, by the Owner(s).

## Warranty

SYBEX warrants the enclosed media to be free of physical defects for a period of ninety (90) days after purchase. The Software is not available from SYBEX in any other form or media than that enclosed herein or posted to *www.sybex.com*. If you discover a defect in the media during this warranty period, you may obtain a replacement of identical format at no charge by sending the defective media, postage prepaid, with proof of purchase to:

SYBEX Inc.
Customer Service Department
1151 Marina Village Parkway
Alameda, CA 94501
(510) 523-8233
Fax: (510) 523-2373
e-mail: info@sybex.com
WEB: HTTP//:WWW.SYBEX.COM

After the 90-day period, you can obtain replacement media of identical format by sending us the defective disk, proof of purchase, and a check or money order for $10, payable to SYBEX.

### Disclaimer

SYBEX makes no warranty or representation, either expressed or implied, with respect to the Software or its contents, quality, performance, merchantability, or fitness for a particular purpose. In no event will SYBEX, its distributors, or dealers be liable to you or any other party for direct, indirect, special, incidental, consequential, or other damages arising out of the use of or inability to use the Software or its contents even if advised of the possibility of such damage. In the event that the Software includes an online update feature, SYBEX further disclaims any obligation to provide this feature for any specific duration other than the initial posting.

The exclusion of implied warranties is not permitted by some states. Therefore, the above exclusion may not apply to you. This warranty provides you with specific legal rights; there may be other rights that you may have that vary from state to state. The pricing of the book with the Software by SYBEX reflects the allocation of risk and limitations on liability contained in this agreement of Terms and Conditions.

## Shareware Distribution

This Software may contain various programs that are distributed as shareware. Copyright laws apply to both shareware and ordinary commercial software, and the copyright Owner(s) retains all rights. If you try a shareware program and continue using it, you are expected to register it. Individual programs differ on details of trial periods, registration, and payment. Please observe the requirements stated in appropriate files.

## Copy Protection

The Software in whole or in part may or may not be copy-protected or encrypted. However, in all cases, reselling or redistributing these files without authorization is expressly forbidden except as specifically provided for by the Owner(s) therein.

**NOTE**

This book is based on CodeWarrior for Windows DR1. The "DR" is an acronym for "Developer Release," which means that the tools are subject to significant change and are not final. If you are working with a later version of CodeWarrior, there are likely to be some discrepancies between the text in this book and the actual product. In all such cases, the actual product clearly takes precedence. If there are significant changes to the product, check Sybex's web site (**http://www.sybex.com**) for possible updates to the contents of this book.

# Disclaimer from the Publisher

The information in this book is subject to the following disclaimers:

- This book describes CodeWarrior for Windows DR1, a developer release that is subject to significant change.
- Metrowerks does not provide support for CodeWarrior Lite, which is included with this book. It does provide support for registered owners of CodeWarrior for Windows DR1.
- Visit Metrowerks Web site at **http://www.metrowerks.com** for additional information on CodeWarrior for Windows.

*This book is dedicated to a select and august group—the hundred or so engineers who, over time and space, have worked to bring CodeWarrior to vibrant life. Most especially, this book is dedicated to the small team that brought CodeWarrior to Windows: Cam Vien, Eric Cloninger, Bob Kushlis, Kurt Ostfeld, Nalinin Prakash, Andrew Southwick, Joel Sumner, and HongGang Zhang. You are, each and all, Code Warriors!*

# ACKNOWLEDGMENTS

One name appears on the cover, but a real team put this book together for you, and each of them deserves special thanks.

Many folks at Metrowerks had a hand directly or indirectly in this book. I'd like to especially thank Greg Galanos and Jean Belanger, the president and chairman of Metrowerks, respectively, for encouragement and support. Dave Mark handled all the arrangements to ensure that CodeWarrior Lite was included with this book. Travis Weller and Brian Auderer helped with the artwork. And of course, thanks to all those amazing engineers to whom this book is dedicated.

Lucien Stavenhagen, from Metrowerks tech support, deserves a real tip of the hat. As tech editor extraordinaire he helped ensure that the text and code in this book are up to date and as close to the leading edge as possible. There's more than one special note about the inner workings of CodeWarrior that Lucien inspired.

Editing brings us to the folks at SYBEX without whom the tome in your hands would not exist. Helping out along the way from manuscript to printed volume were Gary Masters, Suzanne Rotondo, Maureen Adams, LeeAnn Pickrell, and Michael Welch—my principal down-in-the-gears editor. Thanks also go to Amy Eoff for careful proofreading and to Kate Kaminski for creating the pages you see before you. You all had a hand in making me look good, and I really appreciate it. Carole McClendon at Waterside Productions shepherded this project through its very early stages, and a better agent I could not ask for.

Lucien Stavenhagen, Peter Lewis, and Brad Mohr contributed code to demonstrate just how cool CodeWarrior can be. I am proud to have them as my colleagues, and thank them for helping me meet the deadlines.

Most of all, I want to thank Abby Rose, who always believed in this project and who provided technical support of which other authors only dream.

# CONTENTS AT A GLANCE

*Introduction* *xxvi*

**PART I** **Meeting CodeWarrior** **1**

**1** CodeWarrior Overview 3
**2** Installing and Launching CodeWarrior 27

**PART II** **Managing Projects** **43**

**3** Working with Projects 45

**PART III** **Using the Core Tools** **105**

**4** Editing Your Code 107
**5** Searching and Replacing Your Code 159
**6** Browsing Your Code 215
**7** Compiling and Linking Your Code 277
**8** Debugging Your Code 327

**PART IV** **Working with Libraries and Frameworks** **409**

**9** Using Standard Libraries 411
**10** Using Frameworks 439
**11** Using Constructor 463

**PART V** **Hitting the Target** **511**

**12** Programming for Windows 513

**13** Programming for Java 537

**14** Programming for Mac OS 555

**15** CodeWarrior Tomorrow 583

**Appendix A** Contacting Metrowerks 593

**Appendix B** What's Inside CodeWarrior Gold 599

*Index* *606*

# TABLE OF CONTENTS

*Introduction* *xxvi*

**PART I** **Meeting CodeWarrior** **1**

**1 CodeWarrior Overview** **3**
CodeWarrior Core Tools 5
Project Manager 6
Editor 8
Browser 10
Compilers and Linkers 11
Debugger 13
The Integrated Development Environment 14
Frameworks and Advanced Tools 15
Metrowerks Standard Libraries 15
Application Frameworks 17
Constructor 19
Multiple Languages and Targets 20
Compiler Architecture 21
Summary 24

**2 Installing and Launching CodeWarrior** **27**
Installing CodeWarrior 28
Your First Look at CodeWarrior 32
The Menus 32
The Toolbar 33
Hands-On Exercises 36
Summary 40

**PART II** **Managing Projects** **43**

**3 Working with Projects** **45**
Understanding Projects 46
Creating a Project 49
The Project Window 54
Opening a Project 58
Saving a Project 60
Closing a Project 60
File Management in the Project Manager 61
Selecting Files 61
Adding Files 62
Creating and Naming Groups 64
Moving Files and Groups 66
Removing Files and Groups 66
Opening Files 67
Accessing Included Files 69
Cool Features 70
Touching Files 70
Controlling Debugging 71
Setting Project Options 73
Understanding CodeWarrior Options 74
Setting Default Project Options 75
Working with Settings 76
Picking a Target 77
Choosing Project Settings 78
Setting Build Extras 80
Setting Access Paths 81
Customizing Stationery 84
Hands-On Exercises 87
Summary 102

**PART II Using the Core Tools 105**

**4 Editing Your Code 107**
Creating a New File 109
Opening an Existing File 110
Editing Source Code 111
Understanding the Editor Window 112
Adding Text 113
Moving the Insertion Point 114
Selecting Text 117
Moving Text 118
Deleting Text 120
Undoing Changes 120
Choosing Font and Size Preferences 122
Indenting Code 123
Cool Features 124
Splitting Panes 124
Syntax Coloring 126
Balancing Punctuation 129
Jumping to Any Routine 131
Setting Markers 132
Controlling Line Endings 134
Controlling the Look of the Editor 135
Opening Related Files 136
Working with Multiple Windows 137
Saving a File 138
Saving One File 139
Saving Files Automatically 139
Renaming and Saving a File 140
Reverting a File 141
Saving Window Information 141
Closing a File 142
Printing a File 143

| | | |
|---|---|---|
| | Setting Print Options | 143 |
| | Printing a Window | 144 |
| | Hands-On Exercises | 144 |
| | Summary | 157 |
| **5** | **Searching and Replacing Your Code** | **159** |
| | Understanding the Search Tools | 161 |
| | Understanding the Search Design | 161 |
| | Understanding the Search Controls | 162 |
| | Finding Selected Text | 164 |
| | Finding in a Single File | 166 |
| | Entering Search and Replace Text | 166 |
| | Controlling Search Range | 169 |
| | Controlling Search Scope | 170 |
| | Finding and Replacing Text | 171 |
| | Working without the Find Window | 174 |
| | Finding in Multiple Files | 175 |
| | Activating Multi-File Search | 176 |
| | Choosing Files to Be Searched | 176 |
| | Controlling Multi-File Search Range | 180 |
| | Controlling Multi-File Searches | 181 |
| | Working With File Sets | 182 |
| | Cool Features | 185 |
| | Using Batch Reports | 185 |
| | Using Regular Expressions | 191 |
| | Hands-On Exercises | 199 |
| | Summary | 212 |
| **6** | **Browsing Your Code** | **215** |
| | Understanding the Browser | 216 |
| | Activating the Browser | 224 |
| | Accessing the Catalog | 225 |
| | Using the Graphical Browser | 228 |
| | Controlling the Appearance of the Hierarchy Window | 229 |
| | Showing and Hiding Classes | 230 |

Performing Deep and Wide Disclosures 233
Seeing Multiple Inheritance 234
Navigating Code with the Hierarchy View 235
Viewing a Single-Class Hierarchy 238
Using the Class Browser 239
Understanding the Class Browser 241
Viewing and Editing Code with the Class Browser 248
Using the Single-Class Browser 249
Using the Symbol Window 252
Navigating Code with the Browser 254
Changing Browser Views 254
Viewing Class Declarations 255
Viewing Functions 256
Cool Features 257
Going Backward and Forward 258
Setting Browser Options 260
Hands-On Exercises 262
Summary 274

**7 Compiling and Linking Your Code 277**

Choosing a Compiler 278
Plugin Tools 279
Choosing a Target Compiler 280
Choosing a Language Compiler 282
Controlling a Compile 286
Deciding Which Files to Compile 286
Issuing a Compile Command 288
Controlling Code Generation 289
Controlling a Language Compiler 290
Speeding Up a Compile 293
Disassembling Code 297
Preprocessing a C/C++ File 299
Precompiling C/C++ Headers 300

Controlling a Link 303
Choosing a Linker 303
Issuing a Link Command 304
Setting Linker Options 305
Cool Features 307
Understanding Errors and Warnings 307
Setting Compiler Warnings 308
Understanding the Message Window 310
Dealing with Errors 312
Hands-On Exercises 313
Summary 323

**8 Debugging Your Code 327**
Understanding the Debugger 328
Turning On the Debugger 329
Setting Up Debugging Information 332
Launching the Debugger 336
Understanding the Debugger Windows 339
Understanding the Source Browser Window 340
Understanding the Program Window 343
Understanding the Log Window 346
Mastering Basic Debugging 346
Stepping through Code 347
Navigating through Code 355
Moving from Source to Debugger and Back 358
Working with Breakpoints 359
Viewing and Changing Data 367
Saving Your Debug Information 378
Working with Low-Level Debugger Features 380
Debugging Libraries 384
Cool Features 384
Working with Watchpoints 384
Multilanguage Debugging 387
Multitask Debugging 387

Multiproject Debugging 388
Hands-On Exercises 389
Summary 405

**PART IV Working with Libraries and Frameworks 409**

**9 Using Standard Libraries 411**
Using the Standard C Library 414
Understanding the C Library 415
Using the C Library in Your Projects 416
Using the Standard C++ Library 419
Understanding the C++ Library 419
Using the C++ Library in Your Projects 421
Using the Pascal Library 423
Understanding the Pascal Library 423
Using the Pascal Library in Your Projects 425
Cool Features 426
Using Library Source Code 426
Calling MSL C from Pascal 427
Hands-On Exercises 428
Summary 436

**10 Using Frameworks 439**
Using the MFC 443
Using MFC Stationery 443
Including MFC Headers 444
Using MFC Source Code 445
Using Java Packages 447
Using Java Stationery 448
Working with the Classes.zip File 448
Importing Java Classes 450
Using PowerPlant 451
Hands-On Exercises 456
Summary 460

**11 Using Constructor** **463**
Understanding Constructor 465
Working with Constructor 467
Managing Resources 469
Making a New Resource 469
Working with Whole Resources 472
Revisiting the Resource Fork 474
Editing Layouts 476
Using Layouts and Hierarchies 477
Adding a Pane to a Layout 479
Setting Pane Properties 480
Moving and Arranging Panes 481
Editing Menus 486
Making a New Menu 488
Working with the Menu Bar 488
Editing an Individual Menu 490
Adding a Submenu 492
Editing Bitmaps 493
Understanding the Bitmap Editor 494
Using the Bitmap Tools 495
Working with Sample Views 496
Editing Pixels 497
Working with Colors and Patterns 498
Using Transparency 499
Editing Text Traits 500
Cool Features 501
Hands-On Exercises 502
Summary 508

**PART V Hitting the Target** **511**

**12 Programming for Windows** **513**
Setting Up for x86 Development 514
x86 Project Settings 515

x86 CodeGen Settings 517
IR Optimizer Settings 518
x86 Linker Settings 520
WinRC Compiler Settings 522
Other x86 Features 522
Creating a DLL 523
Creating an Application 528
Debugging x86 Code 531
Summary 535

**13 Programming for Java 537**
Setting Up for Java Development 540
Java Project Settings 541
Creating a Java Applet 543
Creating a Java Application 548
Summary 553

**14 Programming for Mac OS 555**
Setting Up for 68K Programming 558
68K Project Settings 558
68K CodeGen Settings 561
68K Linker Settings 563
Setting Up for PowerPC Programming 565
PPC Project Settings 566
PPC Codegen Settings 567
PPC Linker Settings 569
Creating a Mac OS Application 570
Debugging Mac OS Code 578
Summary 579

**15 CodeWarrior Tomorrow 583**
The Path You Traveled 584
Cross-Platform Development 585
Where CodeWarrior Is Going 586
CodeWarrior's New Project Manager 587
The Electronic Revolution 590

**Appendix A Contacting Metrowerks 593**
Metrowerks Web Site 594
Tech Support 594
Sales 595
Registration and Updates 595
The Address 596

**Appendix B What's Inside CodeWarrior Gold 599**
home.htm 600
bin 600
BinSDK 601
CodeWarrior Documentation 602
(Help) 602
(Stationery) 602
CodeWarrior Books 603
CodeWarrior Examples 603
CW Release Notes 603
Java Support 603
Mac OS Support 603
Metrowerks Standard Library 604
Resources 604
Tools 604
Win32-x86 Support 605

*Index* *606*

# INTRODUCTION

CodeWarrior is the world's most powerful and flexible programming environment. CodeWarrior arms you with an arsenal of tools to help you do what you do best: write code. *Mastering CodeWarrior for Windows 95/NT—The Official Guide* shows you how to use these tools.

To start you off, I'd like to take you on a quick tour of CodeWarrior and answer the big question, what is CodeWarrior? After that, we'll look at what this book covers, and how we're going to navigate on our journey through the world of CodeWarrior.

## What Is CodeWarrior?

CodeWarrior is an *integrated development environment*, or *IDE*. What this means is that all the pieces of a classic programming environment—a text editor, compiler, linker, makefiles, debugger, and so forth—are a cohesive unit controlled by a project manager.

IDEs are very common nowadays. The IDE model has been adopted by many programming environments for a good and simple reason: An integrated approach makes you more productive.

CodeWarrior is not just another IDE. What makes CodeWarrior different from other environments—in fact what makes CodeWarrior *unique* in the universe of programming tools—is its unparalleled flexibility. What you can do with CodeWarrior is nothing short of remarkable.

First-generation programming tools were completely unintegrated and difficult to use. They were rather like trying to sail a coastline without a chart.

Second-generation tools invented and applied the IDE concept to a single language and target. This improvement enhanced ease of use and programmer productivity. You not only had a chart, you also had a compass and sextant, and they all worked well together—as long as you were sailing in familiar waters.

A typical second-generation IDE enables you to program in one or at most two closely-related languages (such as C and C++), and write code for one operating

system or computer chip. As a result, if you want to write code in more than one language, or for more than one platform, you must use a whole collection of programming environments. For example, you have one environment for your C/C++ programming for Windows. You have another for Pascal programming for the Mac OS. You have a third for Java programming. You might have a fourth for Pascal programming for Windows, and so on. You have to master each of these tool sets, a nontrivial task.

CodeWarrior, on the other hand, is the integrated development environment of programming environments! What the second-generation IDE concept did for disjointed programming tools, CodeWarrior does in turn for a disjointed collection of programming environments.

CodeWarrior is a third-generation tool set, and takes programming to a whole new level. CodeWarrior is to second-generation programming tools what the global positioning system is to navigation. CodeWarrior is an IDE that supports multiple languages and multiple targets. This means you can use one environment for almost all your programming.

CodeWarrior has a unique modular architecture that supports plug-in compilers and linkers. Consequently, in CodeWarrior you pick the language you want to use, the microprocessor you want to target, and the operating system for which you want to write code.

At present, CodeWarrior supports the following:

- Four languages—C, C++, Object Pascal, and Java
- Four processors—x86 (including Pentium Pro), MIPS, 680x0, and PowerPC
- Nine operating systems—Windows 95, Windows NT, Mac OS, Be OS, Pilot OS, Magic OS, PlayStation OS, PowerTV OS, and the Java Virtual Machine

In addition, CodeWarrior supports assembly language programming for all its target processors.

Although multiple languages, processors, and operating systems in a single environment might seem like programming nirvana, it gets even better. CodeWarrior is available for and runs under Windows 95/NT, Mac OS, and Be OS. The tools running under each of these operating systems are essentially identical. You get the same IDE, the same editor, project manager, debugger, and so forth. So you not only get to pick your language, processor, and operating system, you also get to pick the host platform on which you like to work.

CodeWarrior's language, processor, operating system, and host options present you with a matrix of programming possibilities unmatched by any other tool set. The important and popular combinations of possibilities are all fully supported. You can write code in:

- Any supported language for Windows programming, while running CodeWarrior on either Windows or Mac OS
- Any supported language for Mac OS programming, while running CodeWarrior on either Windows or Mac OS
- Java for the Java Virtual Machine
- C and C++ for the MIPS processor, while running CodeWarrior on either Windows or Mac OS

You should realize, however, that some combinations within this matrix of possibilities aren't currently supported. For example, you cannot write Java code for the Magic hand-held device. You cannot write Be code from Windows-hosted CodeWarrior.

On the other hand, the manufacturer, Metrowerks, adds new targets and compilers regularly, most recently the PowerTV set-top box for interactive television (using the PowerPC compiler), and the PlayStation game console from Sony Computer Entertainment (using the MIPS compiler).

CodeWarrior gives you a framework within which you write and compile code. You decide how to fill in the framework. Because of its plug-in architecture and multiple targets, CodeWarrior is ideal not just for work dedicated to a single platform, but for cross-platform programming as well.

# Overview of This Book

*Mastering CodeWarrior for Windows 95/NT—The Official Guide* covers the CodeWarrior tools in detail. This book shows you how to use CodeWarrior to manage projects, write code, debug code, and program for the Windows OS, the Mac OS, and Java. This book does not teach you the intricacies of programming for any of these operating systems. It concentrates on how to use the tools.

Along the way you're going to encounter a lot of information. I've tried to organize this information in ways that will make the journey enjoyable. The focus lies on the practical knowledge you need to use CodeWarrior, true. However, this book often strays from the cold, dry facts, because a set of tools is more than a collection of windows and menu commands. It is a whole that exceeds these parts.

You'll find sidebar discussions throughout this book. Each sidebar has a title and appears in a shaded box. A sidebar is a side trip on your journey to CodeWarrior mastery. The sidebar presents interesting and significant information related to but not vital to understanding the topics presented in each chapter.

## A Typical Sidebar

Text within the sidebar introduces you to interesting and significant information related to but not essential to understanding the main topic.

For example, to be sure everyone knows what's going on, sometimes the information sidebars present can be pretty fundamental. I've tried to put that information into a series of sidebars called "The Fundamentals." These sidebars present basic information that you may already know. If you are a seasoned programmer, you might want to glance at this information quickly, but if you don't already know it you can probably figure it out quickly on your own.

At the other extreme, it can be very helpful sometimes to know not only how to use the tools, but why the tools are the way they are—that is, what engineering design decisions were made and how they're reflected in the tools. While this material is not purely practical, this kind of high-level insight can be extremely helpful. You'll find this kind of information in a second series of sidebars called "The Thought behind…."

I supply sidebars on various other topics as well, and you'll surely find the information interesting and entertaining. The main text, however, is devoted to the practical realities of windows, menus, accelerators, and options that you use when working with CodeWarrior.

In addition to the sidebars and text, you'll find many notes, tips, and warnings that provide you with an insider's view of the tools, as follows:

**NOTE** Notes cover interesting sidelights or information about a topic.

**TIP** Tips suggest pointers and inside information that can make your programming life easier.

**WARNING** Warnings alert you to subtle or hidden interactions that can cause problems you want to avoid.

This organization of basic, practical, and high-level information extends throughout this book. I've also organized *Mastering CodeWarrior for Windows 95/NT—The Official Guide* linearly into five parts, as follows:

## Part I: Meeting CodeWarrior

This part covers the basic philosophy and architecture of CodeWarrior. In Chapter 1 you meet the individual tools briefly, so you understand what they are and how they work together. In Chapter 2 you install CodeWarrior and get your first look at the real thing.

## Part II: Managing Projects

The second part introduces the IDE through the project manager. This is where you create new projects, add files, and control your work at a high level. Chapter 3 covers all this information.

## Part III: Using the Core Tools

The third part brings you to the tools you work with to create good code. This section is a task-based discussion that focuses on what you do as a programmer, and how CodeWarrior helps you reach your goals quickly and easily. Chapter 4 covers the source code editor and its many advanced tools. Chapter 5 sails through CodeWarrior's powerful search-and-replace engine for code. Chapter 6 discusses the amazing code browser that is functional for both object-based code and procedural code, and that provides tremendous code navigation options. Chapter 7 shows you how to compile and link your code with CodeWarrior. Chapter 8 covers the core debugger.

## Part IV: Working with Libraries and Frameworks

The fourth part discusses CodeWarrior's libraries, frameworks, and rapid application development (RAD) tools. Chapter 9 covers platform-independent standard libraries that come with CodeWarrior. You'll find these libraries useful for C, C++, and Pascal programming. Chapter 10 introduces the three complete frameworks that come with CodeWarrior: MFC (Microsoft Foundation Classes) for Windows 95/NT development, PowerPlant for Mac OS development, and the Java AWT (Abstract Windowing Toolkit) for Java development. Chapter 11 discusses Constructor. Constructor is a forms builder (a visual interface builder).

## Part V: Hitting the Target

The book's final part brings you to your goal: writing real code for a real platform. This section covers the unique requirements of each of the three major target platforms. Chapter 12 discusses the specifics of programming for Windows 95/NT with CodeWarrior. Chapter 13 covers Java programming. Chapter 14 covers Mac OS programming. In Chapter 15, we'll look briefly over the horizon at where CodeWarrior is going in the future.

In addition to the chapters just outlined, Appendix A lists contact information for Metrowerks. Appendix B briefly describes the contents of CodeWarrior Gold for Windows 95/NT—the complete version of Codewarrior for Windows, Java, and Mac OS programming.

## Requirements

*Mastering CodeWarrior for Windows 95/NT—The Official Guide* assumes you are a computer programmer, or that you are interested in programming and are familiar with programming terminology and concepts. If you know what it means to compile and link a program, then you should have no trouble with this book. It does not absolutely require that you know a programming language, or that you know how to program. However, because this book is teaching you how to use a set of powerful programming tools, it is most useful if you have at least beginning programming experience. This book does *not* teach you how to program a computer.

Certain parts of CodeWarrior and this book are especially interesting to advanced and experienced programmers. However, even beginning programmers should find this material accessible and helpful. For example, Chapter 6 discusses the CodeWarrior browser, with a heavy emphasis on object-oriented programming. If you are unfamiliar with object-oriented programming, Chapter 6 has a brief but helpful introduction to the concepts involved. The popularity of object-oriented programming is growing rapidly, and Chapter 6 explains why.

## What You Will Learn

In *Mastering CodeWarrior for Windows 95/NT—The Official Guide* you will learn all about the CodeWarrior tools as they run under Windows 95/NT. You will learn about the following:

- The CodeWarrior architecture
- Installing CodeWarrior
- The project manager
- File management
- The source code editor
- Finding and replacing text
- The source code browser
- Compilers and linkers
- Debugging

- The Metrowerks Standard Libraries
- Application frameworks
- Programming for Java
- Cross-platform programming for the Mac OS

There is a lot to learn, but navigating from here to there will be easier than it might appear. All the CodeWarrior tools share a common interface and design, so learning one gives you a great start towards learning them all.

Although new to the Windows 95/NT host, CodeWarrior has been developing and improving for years. CodeWarrior began on the Macintosh platform, where ease-of-use is not only common but mandatory. This heritage is a real benefit to you.

The demands of Macintosh programming, and the early availability of the first successful graphical user interface, combined to drive programmer tool development on the Mac. For example, the first effective application frameworks appeared for Macintosh programming. Over time, the concepts invented and enhanced on the Mac crossed over to inspire frameworks for other graphical operating systems. Similarly, the first graphical, integrated development environment appeared on the Mac, simply because the operating system could support it.

The history of the Macintosh, from a programmer's perspective, has been one of innovation that brought ease of use and powerful tools to programmers. In a world where power combined with simplicity is the touchstone of success, no success story is greater than CodeWarrior.

## Hands-On Exercises

Learning about CodeWarrior is fun. Each phase gives you hands-on exercises where you work directly with the tools. Let's talk about the exercises for just a moment before we get started, and see how you can use this book most effectively.

Most of the chapters in this book are organized into two principal sections:

- A general discussion about a particular tool and how to use it.
- A hands-on exercise where you follow step-by-step instructions. These steps walk you through common tasks you perform with the tools so you can get familiar and comfortable with them.

Learning about a tool is a two-step process—first you get the theory, and then you get the practice.

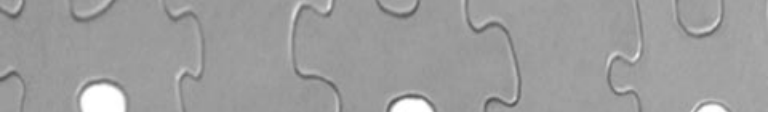
While you are reading the "theory" portion of a chapter, you might want to have your mouse handy all the same. If you're sitting at your computer, you can explore the tools as you read about them for the first time. This can be an excellent introduction to the tools, and can help you understand the text more easily. If something isn't clear to you, you can get your hands on the real thing and see what's going on.

On the other hand, even if you're miles away from your computer while you read the "theory," you don't have to worry about missing something. When you get to the hands-on exercise the steps guide you through each important task you have read about.

This process does have some exceptions. Chapter 1 has no hands-on exercise at all; it is simply an introduction and an overview of the tools. In Chapter 2 you install CodeWarrior, so you should be at your computer to get that done. Finally, Chapters 12 through 14 discuss how to write code for particular platforms; each of these chapters is, in essence, one big hands-on exercise, so you'll want to be at your computer for these chapters.

When you finish this book, you will have mastered the world's most comprehensive and flexible computer programming environment—bar none. That's a pretty exciting prospect, so let's get under way.

# I



# Meeting CodeWarrior

This part of the book introduces you to the CodeWarrior tools and walks you through the CodeWarrior installation process.

In Chapter 1 you learn about the CodeWarrior architecture and the different parts of CodeWarrior. We'll discuss each element briefly so you have a good overview of CodeWarrior's breadth.

Then, in Chapter 2 you install CodeWarrior. CodeWarrior Lite, a limited version of CodeWarrior, is included on the CD accompanying this book. You get your first look at CodeWarrior in this chapter as you launch the tools.

After reading these two chapters, you will be ready to master the tools.

# CHAPTER ONE

1

# CodeWarrior Overview

- CodeWarrior Core Tools
- The Integrated Development Environment
- Frameworks and Advanced Tools
- Multiple Languages and Targets

The first step on the road to mastery is to understand the scope of the tools—what they are, and what they do. You want the big picture.

This chapter gives you a high-level overview of the entire CodeWarrior environment. In subsequent chapters, you will learn about each of these tools in detail, and see how you can use them to accomplish your goals.

Before we begin, let's take a brief look at the terminology you'll encounter throughout this book.

Like any Windows program, CodeWarrior has menus. CodeWarrior also has a toolbar, and of course each menu item has a keyboard equivalent or accelerator. The text in this book often tells you that, to accomplish a certain goal, you should choose a particular item in a menu. You should remember that in all cases you have one or more alternatives to using the mouse to click the menu item.

This book is also very careful in its use of the terms "choose" and "select." The word "select" describes the process of identifying some item (such as a range of text in an editor window) prior to performing some command on that selection. In CodeWarrior, selected items are always highlighted in some fashion. The word "choose" describes deciding which action to perform on the selection.

CodeWarrior is a cross-platform, multilanguage development environment. As a result, we CodeWarriors have to be very careful in how we use programming terms. For example, the term "function" has different meanings for Pascal and C/C++ programmers. With that in mind, Table 1.1 lists some common programming and CodeWarrior-related terms, and explains the way I'll use them in this book.

TABLE 1.1 The use of terms in this book

| Term | Meaning |
|---|---|
| interface file | A file that contains prototypes or descriptions of functions and other declarations. Known in C/C++ as a *header file*. |
| source file | A file that contains code that is compiled, including routine definitions. |
| routine | In C, a *function*. In Pascal, a *function* or *procedure*. Also called a *method* in object-oriented terminology. |
| declaration | Code that describes an item for the compiler. For example, a class description or routine prototype. Does not allocate memory, and generally does not cause the compiler to create any object code. Typically found in an interface file. |
| definition | Code that actually implements a declaration. Generally results in the compiler creating object code. Also known as an *implementation*. Typically found in a source file. |
| host | The operating system under which CodeWarrior is running. |
| target | The designated processor for which you are writing code. |
| process | A running application or code fragment. |

# CodeWarrior Core Tools

The core tools are the principal software components you use to write code. Every development environment, integrated or not, usually consists of one or more of the following individual components:

- Project Manager
- Editor
- Browser
- Compilers and Linkers
- Debugger

Let's take a quick look at each of these components as they are implemented in CodeWarrior.

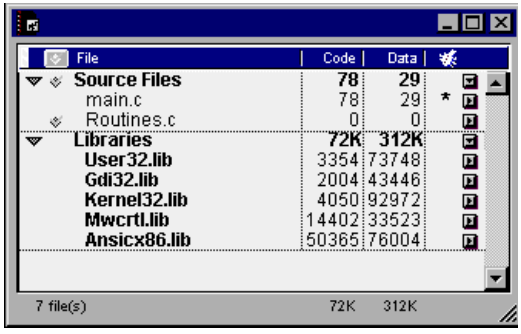
## Project Manager

CodeWarrior is a project-based Integrated Development Environment (IDE). A project consists of a variety of elements. It typically contains interface (header) files, source files, and the corresponding object code. A project has preferences and settings, and may also have pre-compiled libraries of code, resources, notes, and any other file related directly or indirectly to the project. You learn all about the project manager in Chapter 3, "Working with Projects."

To write a program using CodeWarrior, you first need to create a project to track the source files, project preferences, and object code. The project is your control center that brings all the pieces together. The project manager is represented by a single window, as shown in Figure 1.1.

FIGURE 1.1

The project window contains all the files in a project.




This is where you add and remove files from a project, put them into logical groups, control which files are compiled, which will have debugging information created for them, and so forth.

The project manager is more than just a place to collect all the required files. It keeps track of all the interdependencies between files, and keeps track of changes. If one file changes, all files that are dependent upon the changed file are automatically marked for update.

Note what this means: You don't have to keep track of any of this—there is no makefile. The CodeWarrior project manager handles everything for you automatically, freeing you for more productive work.

### The Fundamentals of a Makefile

A makefile is a special file in which you specify exactly which files should be compiled using which compiler. The makefile also specifies how to link the resulting code into a final program. In other words, it is the file that contains the instructions for how to make a program—hence the term *makefile*.

A makefile is very powerful because it allows you to script the entire build process very carefully. Before the advent of the IDE, makefiles were the only game in town.

The downside to makefiles in a nonintegrated development environment is that you have to create them by hand. That means you have to personally keep track of changed files, or rebuild everything every time you want to make the project. You also have to keep track of interfile dependencies, and make sure that necessary files are included before compiling source code.

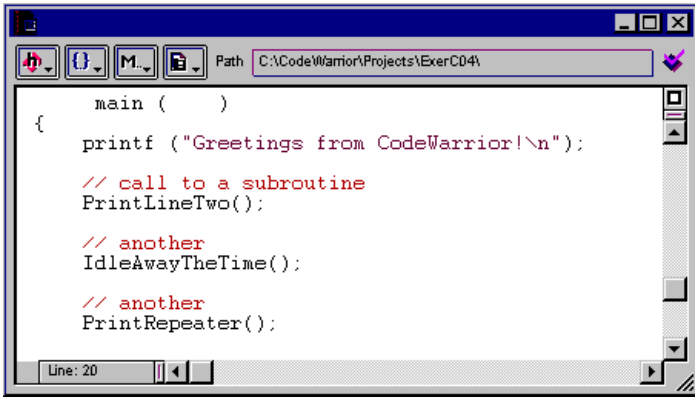
A good IDE such as CodeWarrior takes care of all these details automatically, which makes you more productive.

## Editor

The source code editor is where you write code (see Figure 1.2). An editor window represents the empty slate waiting for your ideas. Chapter 4, "Editing Your Code," covers the editor in greater detail.

The editor has many powerful features. Even in complex projects with hundreds of source files, you can quickly and easily get to where you want to go with a click of the mouse. Your destination may be a different section of the same file, or in any other file in the project.

FIGURE 1.2
You write and edit code in the editor window.




You can use the editor to:

- Split the editor window into multiple panes
- View different parts of the same file simultaneously
- Have multiple windows open simultaneously
- Tile windows automatically so you can see them all
- Toggle between a source file and its related header file
- Open any interface file on which a source file depends
- Locate any function in any file instantly
- Jump to a particular line number
- Set markers at arbitrary locations in any file
- Jump to any marker in a file instantly

This list is only the beginning. The feature list goes on and on. For example, the editor has powerful search and replace features, which are covered in Chapter 5, "Searching and Replacing Your Code."

When you want to look for or replace code, the editor can:

- Find any text
- Replace text manually or automatically in a file
- Search and replace in an arbitrary set of multiple files
- Save sets of files for future searches
- Get a batch report of all successful hits
- Switch between hits to see the related source code
- Perform grep-style searches

**NOTE**

A *grep-style search* uses a special command syntax known as *regular expressions*. Regular expressions allow you to search for strings that include variable text. For example, you can use regular expressions to search for the word "version" when the word is followed by any number less than 4. If you did, the editor would find and accept three different words, version1, version2, and version3.

## Browser

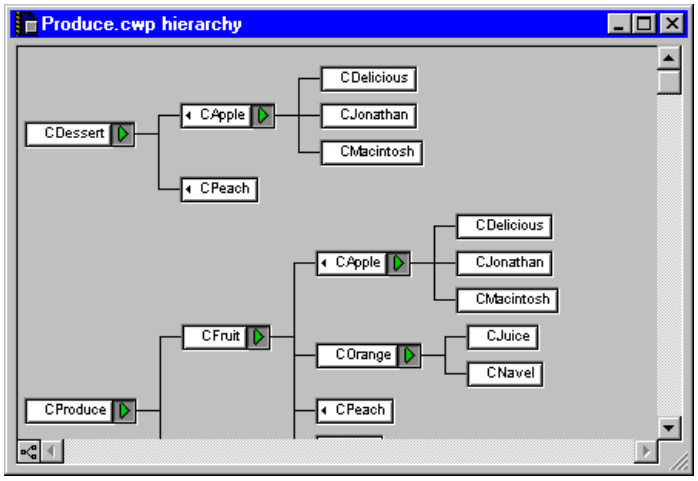
The CodeWarrior graphical browser is without parallel in functionality. Because it is fully integrated into the IDE, all other parts of the environment can take advantage of its features. You'll learn all about the browser in Chapter 6, "Browsing Your Code."

The browser creates a database of all your programming symbols: function names, constants, definitions, enums, classes, typedefs, and more. You can then use a variety of viewing options to study and analyze this data. Figure 1.3 shows one viewing option, the multi-class hierarchy window. This view shows you the hierarchical relationships among classes. It is an elegant and simple representation of complex

class relationships. This view, like the other browser views, is not static. You can use this view as the starting point for a thousand different paths into your code.

FIGURE 1.3

A browser window shows you a complete class hierarchy.



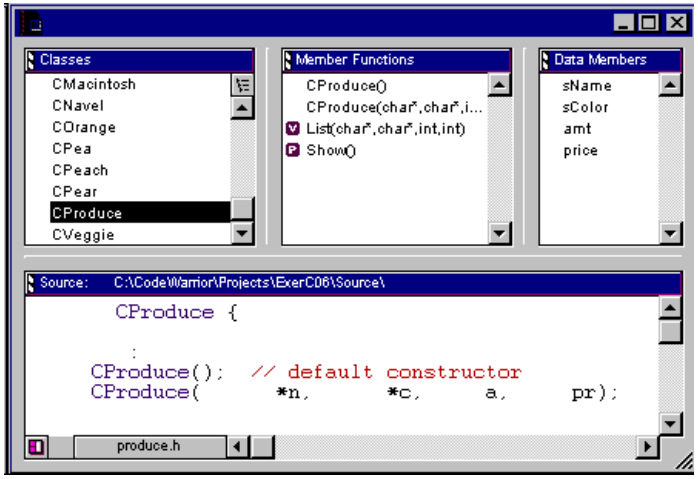

You can also have a more traditional multi-class browser window, as shown in Figure 1.4. In this view the classes appear in the top left pane, the functions in the top center pane, the data members in the top right pane, and source code displays in the bottom pane.

When the browser is active, it remembers every place you have been in CodeWarrior—what you were looking at, what files were open, and where you were in each file. In other words, the browser remembers the state of the IDE. You can sail back and forth through each different location at will. You can go back to where you were, even if it is in a window that you just closed. So, when you are looking for something, and you know you just saw it but you can't find it again, now you can.

Although some of the hottest features such as the class hierarchy view apply only to object-based code, many of the browser's database features apply equally to procedural code.

FIGURE 1.4

The multi-class browser window shows you all the classes and their members.




## Compilers and Linkers

You tell CodeWarrior what compiler to use for each kind of file. You specify the compilers in the Target settings dialog box, as shown in Figure 1.5. Chapter 7, “Compiling and Linking Your Code,” tells you all about it.

The process of compiling and linking is almost completely transparent. You issue the command, and CodeWarrior does the rest. The only time it isn’t transparent is when there’s a problem. CodeWarrior compilers and linkers generate warnings and errors, over which you have complete control. Warnings and errors appear in a special window, like that shown in Figure 1.6.

You can walk through the list of errors, see the source code right in the message window, and fix it on the spot. No one likes errors, but at least you get to find and fix them easily.

FIGURE 1.5

Choose a compiler to work with your code.

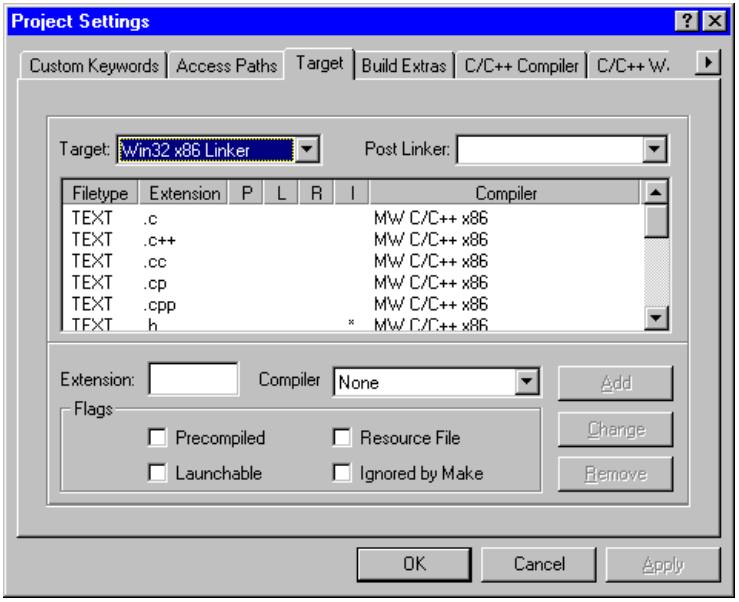
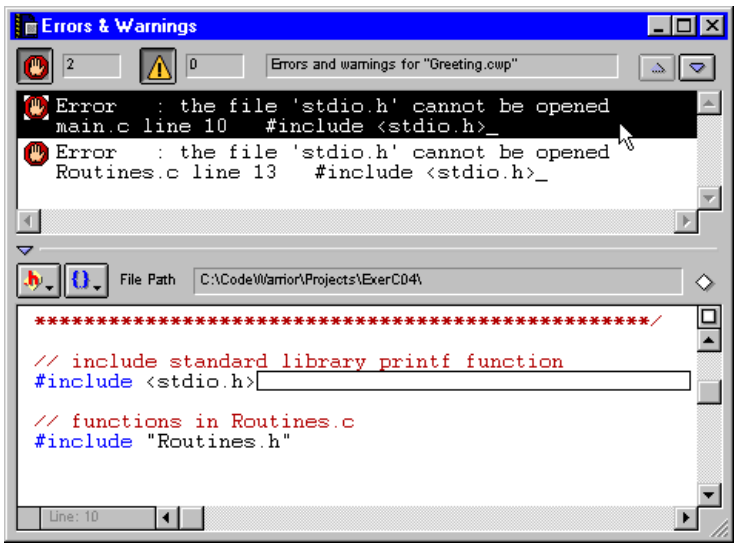



FIGURE 1.6

The message window lists all compiler errors and warnings.



CodeWarrior does not limit you to Metrowerks-provided compilers. CodeWarrior has a modular, plug-in architecture for compilers and linkers. Metrowerks has a public Application Programming Interface (API) for this architecture so third parties can create CodeWarrior-compatible versions of their compilers. Several such compilers are available on the market for the Mac-hosted version of CodeWarrior, when targeting the Mac OS.

### Debugger

The last critical element of a programming tool set is the debugger. In addition to being a powerful tool with all the usual features, the CodeWarrior debugger works with all kinds of code in all the supported languages—C, C++, Object Pascal, and Java. You can debug libraries, executables, multiple threads, and processes. It also supports debugging code running on multiple processors in the same machine.

Figure 1.7 shows the window that appears when you are debugging a running process. You have many ways to view data and control program flow. Chapter 8, "Debugging Your Code," provides all the information on the debugger.

## The Integrated Development Environment

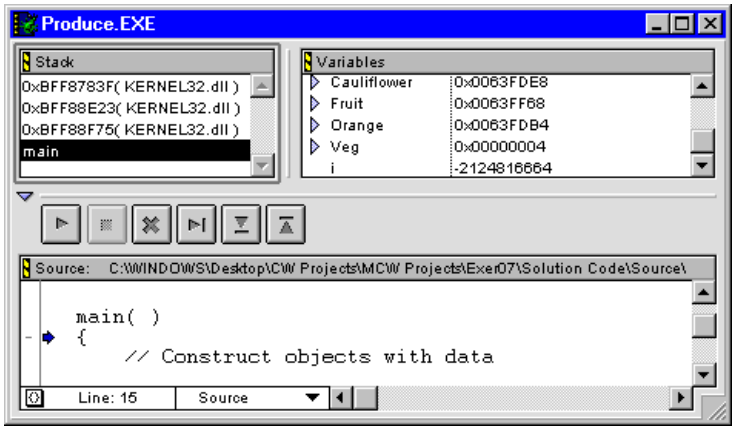
An integrated development environment gathers all the disparate tools you need to go from concept to code, and puts them together under one roof. CodeWarrior is no exception.

In CodeWarrior, the project manager, editor, and browser are all contained in a single application. You can move back and forth among them seamlessly.

The compilers and linkers are separate plug-in modules, managed by the IDE. You never have to launch them separately, locate them,

FIGURE 1.7

The debugger lets you step through code.




open them, or set them. You control all compiler settings and operations from within the IDE. Once again, the process is completely seamless and works in exactly the same way for all compilers and linkers. Learn one compiler, and you've learned them all.

The debugger is the only major component that is not fully integrated into the IDE—it exists as a separate application. Although the debugger application is separate, you'll learn how to launch it and debug code from within the IDE. You use the IDE to decide which files have debugging information created for them, and whether to launch the debugger and work with your code under debugger control.

This separation of the debugger from the IDE lets you use the debugger as a completely stand-alone process. You can open, control, and debug a compiled application without using the rest of CodeWarrior. This can be extremely useful if your computer is short on memory.

# Frameworks and Advanced Tools

You have enough challenges without reinventing the programming wheel for every project. A good development environment includes code and tools you can use to get started on your projects. CodeWarrior provides you with three crucial items:

1. Standard libraries
2. Application frameworks
3. Constructor

Let's examine each of them in turn.

## Metrowerks Standard Libraries

No programming environment would be complete without providing you with standard generic code. This code provides routines for common tasks such as data input and output, memory management, string manipulation, and much more. As a programmer, it is likely you have used standard library code extensively.

Metrowerks provides standard libraries for C, C++, and Pascal. Together these are known as the Metrowerks Standard Libraries, or the MSL. Chapter 9, "Working with the Metrowerks Standard Libraries," shows you how to use these libraries in a CodeWarrior project. The MSL provides you with all the fundamental, generic code you need.

## Application Frameworks

Programming for a modern, graphical operating system such as Windows 95/NT is nontrivial. The code overhead required to manage events, display windows, and handle window content is significant.

## The Fundamentals of a Code Library

A code library is a collection of routines you can use in your own code. Such a library gives you code that you can reuse in project after project. It also saves you from wasting significant effort solving problems that have already been solved.

For example, none of the programming languages supported in CodeWarrior has a cosine command to calculate the cosine of an angle. The word *cosine* is not part of the language syntax.

I think it's a safe bet that you do not want to figure out how to calculate a cosine mathematically. Even if you did want to undertake this task, write the code, and then thoroughly test your algorithm, what guarantee would you have that you have written a fast routine?

Of course, a smarter solution is to use someone else's code. You will find it in a code library. If you shop around, you can find all kinds of code libraries designed for all kinds of purposes, including mathematical libraries, astronomical libraries, financial calculation libraries, and so forth. By collecting good libraries, you give yourself the best possible tools so you can be more productive.

For some languages, most notably C and C++, standard libraries have been defined. These libraries include the most common routines you want to use. They perform mathematical calculations, string manipulation, memory management, data input and output, and more.

Fortunately, the kind of code you have to write to handle these chores can be and has been standardized.

The development of application frameworks over the past few years has been driven by the difficulties of graphical OS programming. The existence of frameworks has greatly simplified the task of programming for graphical operating systems.

CodeWarrior comes with three complete application frameworks, one for each of the major CodeWarrior targets, as follows:

- Microsoft Foundation Classes (MFC) for Windows 95/NT, from Microsoft
- Abstract Windowing Toolkit (AWT) for Java programming, from Sun Microsystems
- PowerPlant for Mac OS programming, from Metrowerks

Chapter 10, "Using Frameworks," explores how to use each of these frameworks.

### The Fundamentals of an Application Framework

An application framework is a structured collection of code designed to provide you with a basic, empty application. Without modification, a framework typically gives you the ability to easily create a simple application empty of any content.

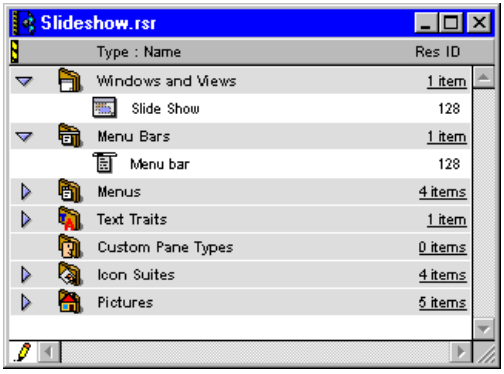
continued ➠

## Constructor

Constructor is an interface builder with which you can quickly and easily define the entire visual interface for an application. You use Constructor to design the appearance and content of windows and dialog boxes, to create menus, and more. Figure 1.8 shows a Constructor project window. Chapter 11, "Using Constructor," covers Constructor in detail.

Constructor is a separate application, and not integrated into the CodeWarrior IDE. However, it follows the same project model as the

FIGURE 1.8
Constructor uses the same project concept and displays all resources in one window.




IDE, and has a similar interface, so learning how to use Constructor is easy.

Constructor has many excellent features. It maintains a catalog of visible interface elements that you can simply drag and drop into a window. You can resize and relocate an item interactively by dragging the item with the mouse. You can even design menus and menu bars, moving menu items around by dragging and dropping. You can create custom text definitions, bitmaps, and custom interface elements.

Constructor is useful when you target the Mac OS using the PowerPlant framework. It does not yet support Windows 95/NT visual interface objects. We'll discuss the future of Constructor for Windows programming in Chapter 11.

## Multiple Languages and Targets

The one major topic we haven't covered yet is how you use these tools to write code for particular targets, such as Windows 95/NT, Java, or the Mac OS. This book has a chapter for each of these targets. These chapters discuss the libraries you include, how to set up the IDE prop-

## The Fundamentals of an Application Framework

Frameworks are usually written in an object-oriented language because of the superior flexibility and extensibility of the object-oriented approach to programming. As a result, any given framework is usually a structured collection of classes. You derive your own custom classes from those provided in the framework. Your custom classes give your application its unique content and interface.

In general, an application framework concerns itself with the application interface. In modern graphical operating systems, standard interface elements have a precise look and behavior, such as scroll bars, check boxes, menus, and so on. All applications written for the operating system use the same visual elements. The framework provides you with boilerplate code to implement these standard objects.

Application frameworks generally provide less support for content. This is because the data content of a particular application is much less predictable. You can use the same interface elements for a word processor and a graphics program. However, the data in those two programs differs greatly.

Nevertheless, a framework can and should provide basic support for common data structures such as arrays, linked lists, tables, and so forth. You can also get support for such data structures from a C++ standard library, such as the Metrowerks Standard Library.

The advantages of a powerful application framework are immediately obvious. With very little effort, you have a complete application, ready to be customized for your own purpose.

For example, a framework is likely to have a **Draw()** behavior or method. You'll find this behavior as part of any visible interface element. You subclass from a framework class, and then override the **Draw()** behavior so your object draws correctly.

continued ➠

### The Fundamentals of an Application Framework

You don't have to worry about when to draw, only about how to draw. The framework takes care of "when" and calls your object's **Draw()** function at the appropriate times.

Solutions for common problems such as event handling, menu handling, command handling, idle-time processing, and thread management are likely to be included in a good framework. Other programmers have already solved these basic problems, and their solutions are embedded in the framework code. You have better things to do with your time than reinvent this wheel. Like a code library, a framework gives you basic solutions for free. You save yourself significant time and effort.

erly for the target, what kind of stationery (template) project to use, how to debug code for that target, and much more.

However, one more high-level insight would be useful to have before you plunge into the tools themselves. You might want to know some of the background behind CodeWarrior's ability to work with multiple languages and targets. This flexibility is the result of intelligent compiler design.

## Compiler Architecture

How does a compiler generate code? Fundamentally, it is a two-step process. First, the compiler reads the text in the source code, and figures out what you are trying to accomplish. Then it turns that interpretation into the appropriate instructions for the microprocessor for which the compiler is designed.

Let's call these two steps the *front-end* and *back-end* tasks of the compiler. The front-end task is language interpretation. The back-end task is code generation.

In a typical compiler, the front-end and back-end tasks are inextricably linked. The compiler is designed to understand one language, and to interpret code in that language into the proper operating instructions for a single processor. As a result, a single compiler is limited to working with a single language and a single target.

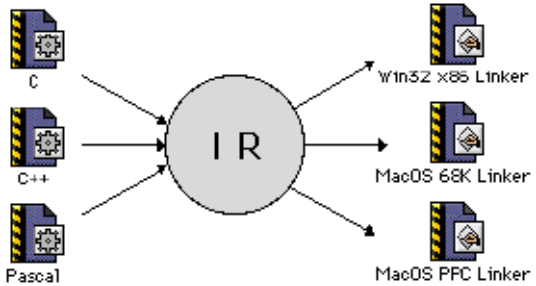
Metrowerks' compiler engineers separated the two tasks, and added a level of abstraction to the compiling process. When you compile code in CodeWarrior, it goes through two separate compilers: a language-oriented front-end compiler, and a processor-oriented back-end compiler (see Figure 1.9). These two compilers are connected by a proprietary technology known as the intermediate representation, or the IR. A CodeWarrior compiler does not interpret a language directly into processor-specific instructions. Instead, it interprets the source code into the IR.

Each language supported by CodeWarrior—C/C++, Pascal, and Java—has a front-end compiler. Because each interprets into the IR, each language front-end is (in theory) processor-blind. It needs to know nothing about the processor for which the code is destined.

A back-end compiler generates code for each target processor. The back-end compiler works from the IR—the intermediate representation. As a result, the back-end is (in theory) language-blind. It can

FIGURE 1.9

The CodeWarrior compiler architecture separates front-end language from back-end target.



compile IR code for the target regardless of the original language used to write the code.

The effort required to implement the elegant concept of an intermediate representation is hardly trivial, to say the least. The process is not perfect, and reality and theory do not always come together painlessly. You can imagine the problems that might arise.

**NOTE**

Actually, the Java compiler doesn't interpret into the IR just yet. It compiles directly to Java bytecodes, so the Java compiler is more like a typical compiler than a CodeWarrior front-end compiler. So, in CodeWarrior you can use Java language to program for one target, the Java Virtual Machine (JVM).

The Java Virtual Machine presents a good example. In certain respects, the Java language is a feature-limited environment. For example, Java has no multiple inheritance, and has no free and direct access to memory. Consequently, how do you interpret code for the Java Virtual Machine when that code comes from a feature-rich language that allows these illegal (in Java) operations?

No easy answers solve these problems. That's why right now in CodeWarrior, if you want to program for the JVM, you write your code in Java. You can't write C++ code and compile it into Java bytecodes—at least, not yet.

## What Is a Virtual Machine?

A "real" microprocessor of silicon has a unique instruction set. A compiler (in Metrowerks architecture, a back-end compiler) generates code based on the instruction set to tell the chip what to do.

Although each chip has a unique instruction set, the instruction sets overlap significantly from one family of chips to another. All central processors do more or less the same thing, so they have very similar instructions.

The Java Virtual Machine is a generic instruction set designed to mimic the most common microprocessors. In other words, the designers of the Java Virtual Machine said, "If we had a real chip, these are the instructions it would use," and they made sure that their instruction set was as generic and universal as possible.

Keep in mind, the JVM is a real instruction set, and it's not at all virtual. The instruction set is designed for a virtual machine. You write source code in the Java language, and a compiler generates JVM instructions. This is conceptually the same thing as writing code in C and compiling for a Pentium chip.

The power of the JVM shows up at runtime. When you run JVM instructions on a "real" chip such as a Pentium or a PowerPC chip, those instructions are translated into processor-specific instructions. Therefore, JVM code is platform-independent. The same code should run in the same way on any platform that has a JVM translator.

Programming-tool vendors and operating system engineers are the folks responsible for writing the code that does the translation from the Java Virtual Machine instruction set to the processor-specific instruction set. This might be a runtime interpreter, or a just-in-time compiler. Either way, the translation should be automatic and reliable on any platform.

Despite these challenges and hurdles, the invention and implementation of the IR has allowed Metrowerks to create a remarkably flexible programming environment where, within certain limits, you are free to pick your language, processor, and operating system in any combination.

## Summary

This completes our quick tour through the CodeWarrior tool set. In this chapter you met all the major components of CodeWarrior. You have a high-level overview of both core tools and advanced tools, and you've seen a glimpse of how they work together.

The CodeWarrior IDE encompasses the project manager, the source code editor, the browser, and compilers and linkers. The IDE controls all compilers and linkers, although the compilers and linkers are in fact separate plug-in components.

CodeWarrior also includes a full-featured source-level debugger that you can launch from within the IDE. The debugger is not as tightly integrated as the other components of the CodeWarrior core tools, so you can use it independently if you wish.

You also met the advanced programming tools that come with CodeWarrior: the Metrowerks standard libraries, application frameworks, and Constructor.

In addition, you have seen the elegant idea that lies at the core of CodeWarrior's remarkable flexibility and success—the intermediate representation. The IR separates languages from processors, and allows Metrowerks to leverage its research and development investment to support multiple languages and targets.

As you work with CodeWarrior, you will never see the IR, although it is there all the same. It is the conceptual underpinning, the keel of the entire tool set. You can see the effects of the IR in the CodeWarrior IDE itself. The signs are subtle, but they are there, and I'll point them out when we encounter them in our explorations.

# CHAPTER TWO

2

# Installing and Launching CodeWarrior

- Installing CodeWarrior
- Your First Look at CodeWarrior
- The Menus
- The Toolbar
- Hands-On Exercises

This chapter shows you how to install and launch CodeWarrior. In addition, you'll get your first serious look at the CodeWarrior IDE, including its menus and a toolbar that contains the most common commands you use in CodeWarrior, such as creating a new file, setting preferences, and many more.

As the introduction mentioned, some of the material we're going to cover is pretty basic. Even though a good deal of this information is segregated throughout the book into "Fundamentals" sidebars, you'll still encounter some very basic concepts. For example, in this chapter you'll meet a toolbar. Now, a toolbar is hardly a brilliant innovation—you've seen them in lots of different programs. Nevertheless, you might want to take a few moments to read about how CodeWarrior uses its toolbar, just to ensure that you're familiar with how it works.

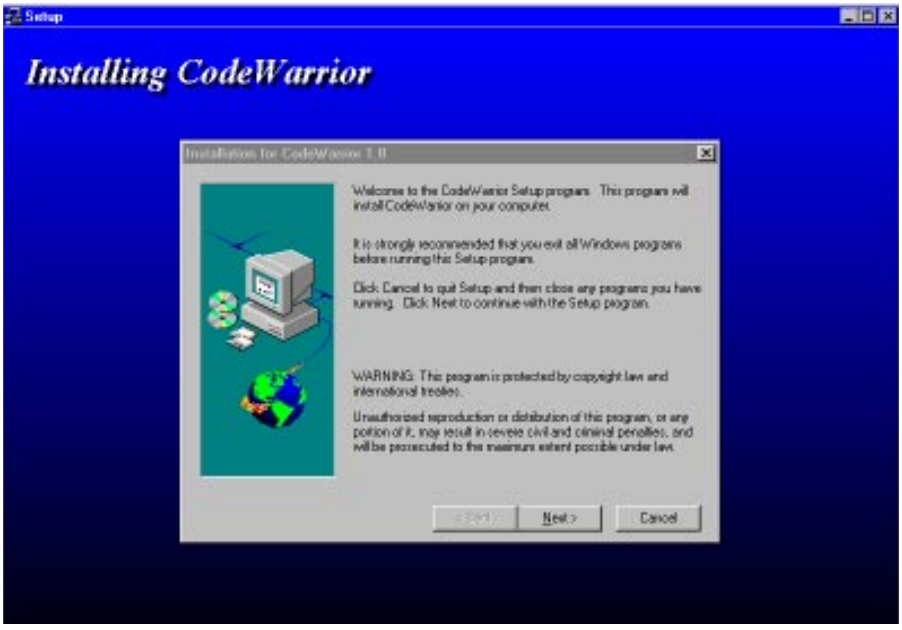
## Installing CodeWarrior

Your first step to launching CodeWarrior is to put the CD-ROM for this book into your CD-ROM drive, and locate the CodeWarrior Lite Setup application. Double-click the icon for the installer, and you are on your way. The installer does some preliminary setup work automatically (decompressing its own files), and then presents you with the opening screen, as shown in Figure 2.1.

**NOTE**

The actual screens you see may vary somewhat from these figures, because I'm using the installer for CodeWarrior itself, the full package. You may be installing CodeWarrior Lite from the CD-ROM accompanying this book. The installing process you encounter should be very similar to what's discussed here.

FIGURE 2.1
The CodeWarrior installer appears when you launch CodeWarrior Lite Installer.

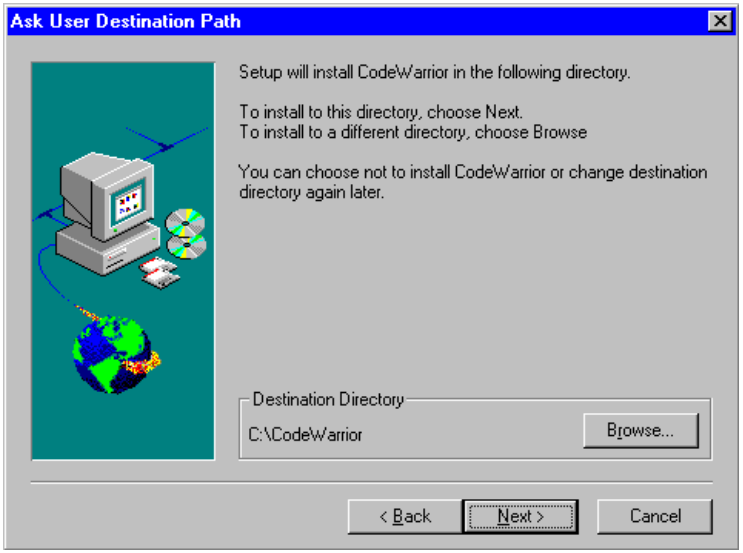

Setup
Installing CodeWarrior
Welcome to the CodeWarrior Setup program. This program will install CodeWarrior on your computer.
It is strongly recommended that you exit all Windows programs before running this Setup program.
Click Cancel to quit Setup and then close any programs you have running. Click Next to continue with the Setup program.
WARNING: This program is protected by copyright law and international treaties.
Unauthorized reproduction or distribution of this program, or any portion of it, may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under law.
Next >
Cancel

Click the Next button, and you should see the CodeWarrior license agreement. Read this agreement, and click Yes if you accept the terms and want to continue installing CodeWarrior. At this point you may be given an opportunity to choose which parts of CodeWarrior to install. If so, make sure you install all the components—the tools, support files, and code for this book. You may also want to install the CodeWarrior documentation that comes on the CD-ROM.

Next, Setup asks you for the folder where you want CodeWarrior installed, as shown in Figure 2.2. You can simply accept the default suggestion by clicking the Next button, or specify a new directory (to do so, click Browse). Whatever path you choose, make sure you remember what it is, because that's where you'll have to go to look for CodeWarrior and its various files. Then click the Next button to continue.

Setup installs shortcuts in the Windows Start menu for various CodeWarrior components such as the IDE and debugger. In the next

FIGURE 2.2
Tell the installer where you want to put CodeWarrior.


Ask User Destination Path
Setup will install CodeWarrior in the following directory.
To install to this directory, choose Next.
To install to a different directory, choose Browse
You can choose not to install CodeWarrior or change destination directory again later.
Destination Directory
C:\CodeWarrior
Browse...
< Back
Next >
Cancel

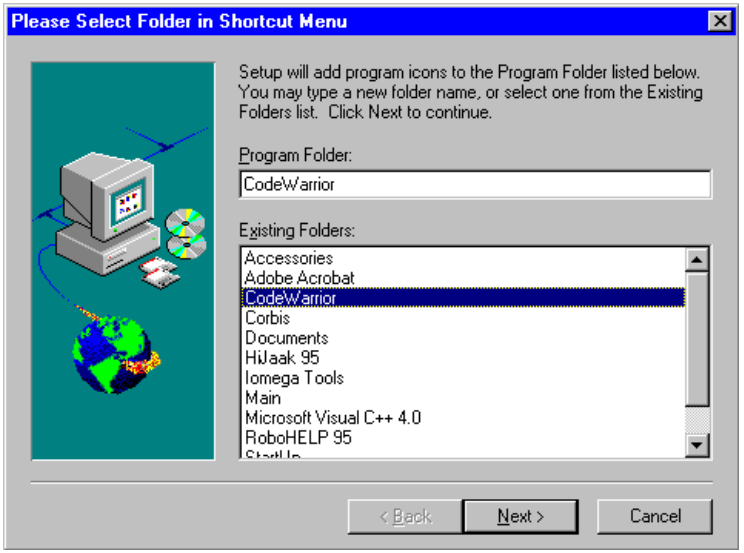
screen, shown in Figure 2.3, you tell Setup what folder inside the Start menu's Program folder you want to use for the CodeWarrior shortcuts. Once again, the default suggestion should work just fine. Click Next after you have specified your shortcut folder.

The installer then tells you what it is going to install (typically a full installation of CodeWarrior) and where, based on your choices. Review the list to make sure it is acceptable. If it is not, return to previous screens to modify the Setup options. If it is acceptable, click Next to continue.

Setup then installs all the files necessary to run CodeWarrior. You'll see active progress indicators the whole way. When the installer is through copying files, you may be presented with more opportunities to install software. For example, Setup offers to install Adobe Acrobat Reader for you, because that's the utility used to read CodeWarrior's electronic documentation. If you do not have Acrobat Reader already, you should choose to install it. CodeWarrior comes with thousands of pages of well-structured documentation, but without Acrobat Reader you can't use it.

FIGURE 2.3

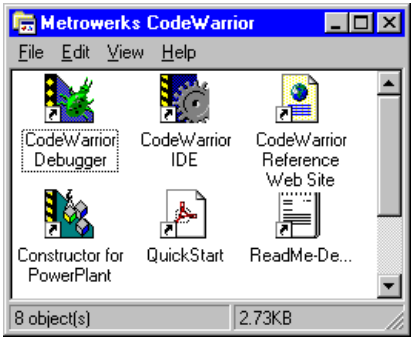
You can install program icons as part of installing CodeWarrior.




When you are all through, click OK to exit Setup and return to the Windows desktop. You'll see an open window similar to that shown in Figure 2.4. This window is the CodeWarrior program folder that appears in the Windows Start menu. It has a shortcut to the CodeWarrior IDE application and the debugger, among other files.

FIGURE 2.4

When the installation is complete, the installer displays the CodeWarrior program folder.

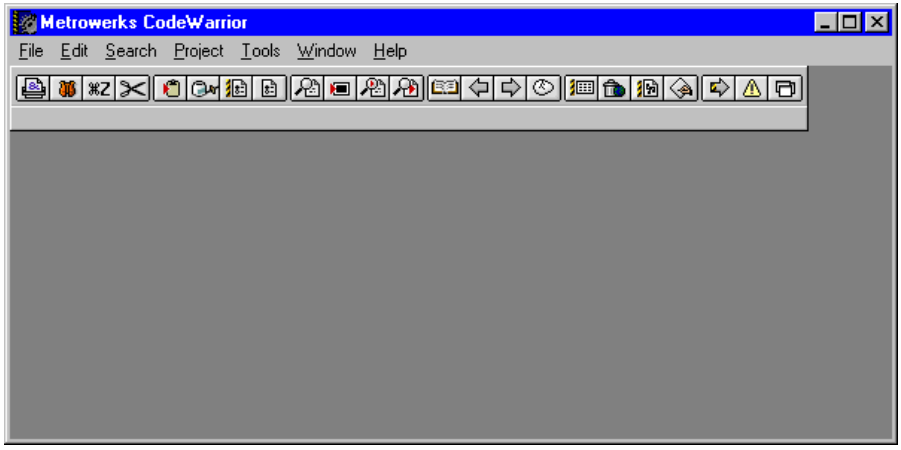



You're ready to go. Simply double-click the IDE shortcut icon to launch CodeWarrior.

## Your First Look at CodeWarrior

When CodeWarrior launches, you'll see the CodeWarrior multiple document interface (MDI) window, as shown in Figure 2.5. The CodeWarrior menus appear along the top of the MDI window, just as you would expect. In all respects, this is a standard MDI window.

FIGURE 2.5
The MDI window is your CodeWarrior workspace.




A toolbar appears near the top of the window as well. Let's examine the CodeWarrior menus briefly, and the toolbar in detail.

## The Menus

The menu bar across the top of the MDI window contains all the commands you use to control CodeWarrior. This includes commands to create a new project, compile your code, disassemble a file, search

for text, and so forth. As you can see from Figure 2.5, you have the following menus:

- **File:** commands to manage files, such as opening, closing, and printing
- **Edit:** commands to cut, copy, and paste, and to set CodeWarrior options
- **Search:** commands to search for and replace text
- **Project:** commands to manage, compile, and build a project
- **Tools:** commands to control the toolbar
- **Window:** commands to manage and display windows, including the browser
- **Help:** commands to access online documentation

We won't discuss the particular items in most of these menus at this time. Over the course of this book you'll learn all about these menus and each command in them. However, we will talk all about the Tools menu now, because it controls the toolbar, which is one of the first things you see in CodeWarrior. The toolbar plays an important role in everything you do with the CodeWarrior tools. The toolbar gives you one-click access to the most common tasks in CodeWarrior.

## The Toolbar

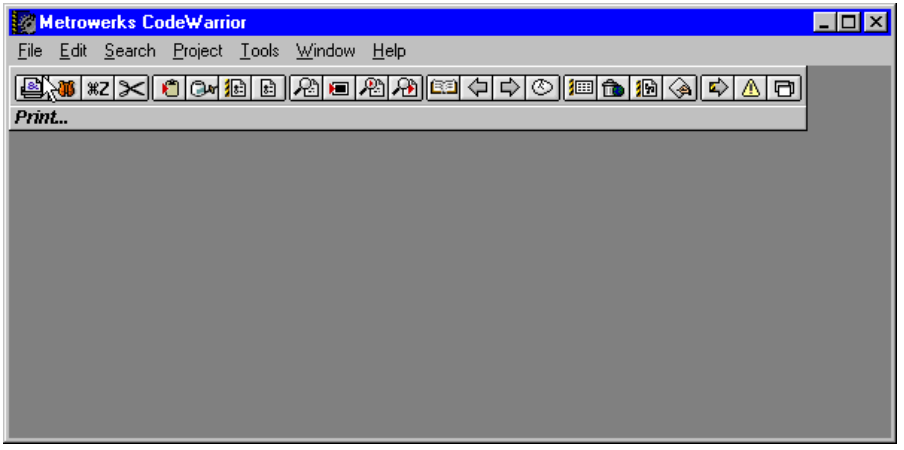
The toolbar displays a collection of icons that correspond to menu items. If you click the icon for a particular command, the result is the same as if you chose it from the menu. The toolbar gives you direct, single-click access to a set of common CodeWarrior actions.

Unless you hide it, the toolbar is available at all times in the IDE. Throughout this book, when I refer to a particular menu item, you should remember that you may be able to use the toolbar to accomplish the same purpose with a single click.

The CodeWarrior engineers know that it can be difficult to remember which icon represents which command. A miniature built-in help system can assist you in remembering. Here's how it works: While the pointer is over a particular icon, the name of the corresponding command appears in the message area beneath the toolbar. For example, in Figure 2.6 the pointer is over the icon for the Print command. The toolbar has a set of the most commonly used commands, such as print, cut, copy, paste, and so forth. If a command is temporarily disabled, its name appears in italics.

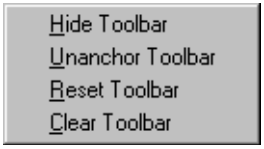
FIGURE 2.6

The CodeWarrior toolbar identifies its own icons.




Unlike some toolbars, this one is movable. You can unanchor the toolbar and move it anywhere you want. When anchored, you see it at its normal place just under the menu bar. When unanchored, the toolbar becomes a floating palette you can move anywhere you like in the CodeWarrior workspace for the most convenient access while you work. The commands to anchor and unanchor the toolbar appear in the Tools menu, as shown in Figure 2.7.

FIGURE 2.7
Use the CodeWarrior Tools menu to manage the toolbar.




**TIP**

CodeWarrior is very smart about menus. The text in a menu item may change depending upon circumstances. For instance, in Figure 2.7 the Unanchor Toolbar item is highlighted because the toolbar is currently anchored. If you unanchor the toolbar, the next time you look in this menu that item will read "Anchor Toolbar." You always get a clear message about what you can do at that moment, based on the state of the IDE. Similarly, menu items become disabled automatically when they are not available

As you can see from the menu, you can control other common properties of the toolbar. The Tools menu lets you show, hide, anchor, unanchor, clear, and reset the toolbar. Clearing the toolbar erases all the icons, but leaves the message visible. Resetting the toolbar restores all the icons.

**NOTE**

CodeWarrior uses the message area of the toolbar to display status messages when it compiles a file. We'll talk about compiling in Chapter 7.

The toolbar is a useful mechanism for controlling the IDE. Some programmers really enjoy the simplicity of the one-click interface. Others prefer the keyboard commands, or the menu interface. In CodeWarrior, the choice is yours. Let's see how it works.

## Hands-On Exercises

This is the first exercise, so it will be very simple. However, these exercises are carefully structured to make them as easy and effective as possible. A brief explanation of how a typical exercise is put together will help you sail through them quickly.

Each exercise begins with a brief overview of what you're going to accomplish. You'll quickly know what you're going to learn. Then you'll see a series of numbered steps. Follow the directions in each step carefully, especially if the step involves writing code. You won't see many steps that require you to write code, though, because most of the time you'll be focusing on the tools. I'll give you all the code you need, except in those places where you're learning how to use CodeWarrior to create and edit your own code.

On some occasions the steps are order-dependent, so you should follow along carefully. I've designed these steps so you can practice each and every fundamental skill necessary to become a code warrior. So, let's get started with the first exercise.

What you'll do in this exercise is launch CodeWarrior, look at the menus, and play with the toolbar to see how it works. We'll start on some real work in the next chapter, where you will create a new project, add files to it, and much more.

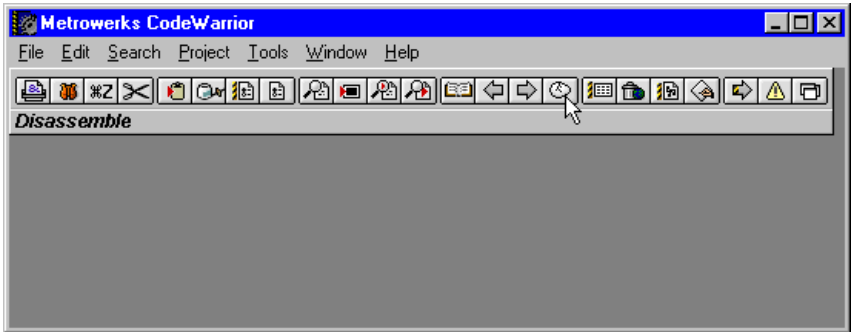
**1. Launch CodeWarrior.** If CodeWarrior isn't already running on your computer, use whatever method you prefer to launch the IDE. For example, you can use the Windows Start menu, or locate the icon for the CodeWarrior IDE, and double-click the icon.

Whatever technique you use, the IDE should launch, and the CodeWarrior MDI window should appear, as shown in Figure 2.8.

**2. Examine the menus.** Click each menu to examine its contents. For example, look in the File menu to see the things you can do with files. The File menu has items you use to open and close projects and files, save and revert, print, and exit from the program.

FIGURE 2.8

The CodeWarrior menus appear at the top of the MDI window.




Examine the other menus as well. Notice the kinds of commands that exist in each one. You don't have to memorize the location of each command. I'll tell you exactly where each item is as you encounter it in the text. When you become familiar with CodeWarrior you'll know where these items are automatically.

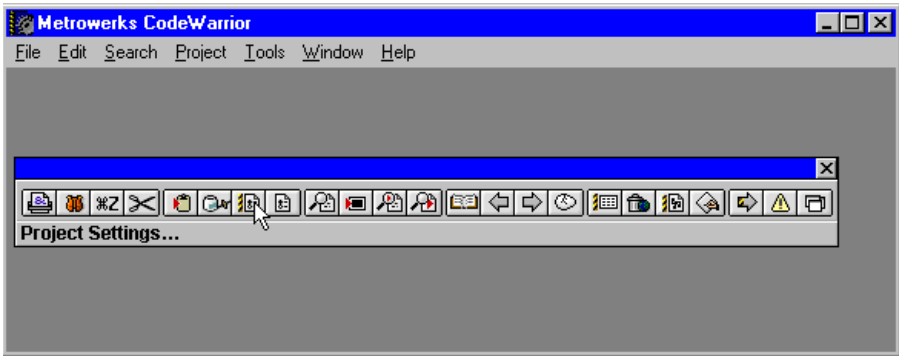
For now, just get a sense for how the menus group items together—the menu titles make the organization pretty obvious. For example, file-related items appear in the File menu. Commands that control your projects cluster in the Project menu.

**3. Examine the toolbar.** Make sure the toolbar is visible. If it isn't, choose the Show Toolbar item in the Tools menu to make it visible. Then move the cursor over each icon to see what it does. Watch the messages in the status part of the toolbar. You'll see that the toolbar has icons for a variety of common tasks, such as cutting, copying, and pasting text.

As you examine the different icons, notice that the names match items in various menus. For example, you can disassemble by clicking the Disassemble icon or by choosing the Disassemble item in the Project menu.

**4. Unanchor the toolbar.** Choose the Unanchor Toolbar item from the Tools menu. The toolbar detaches from the menu bar and becomes a separate window, as shown in Figure 2.9.

FIGURE 2.9
The toolbar is a floating palette when unanchored.




**TIP**

If the descriptive text in the status area is in *italics*, that particular command is currently unavailable. For example, the Print item appears in italics when no file is open for you to print.

Move the toolbar around and play with it. Feel free to anchor it again when you're done if you prefer it locked in place.

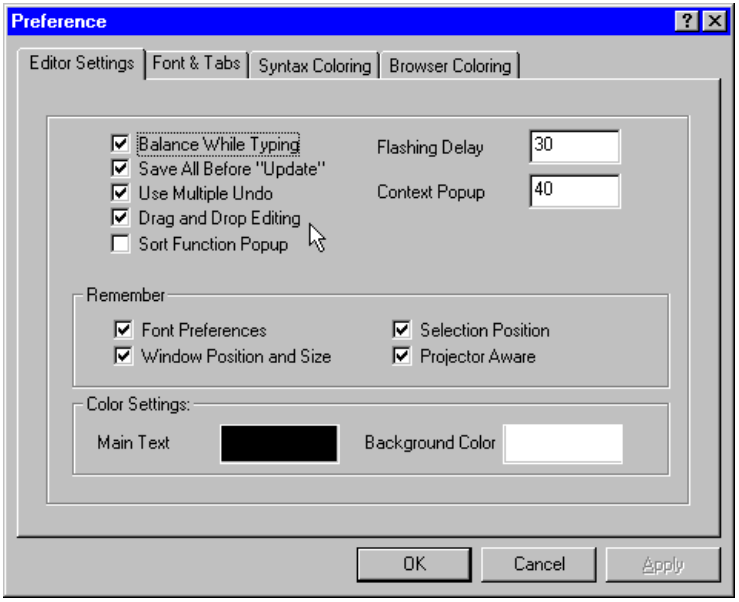
**5. Issue a command from the toolbar.** Click the Preferences icon in the toolbar, and a dialog box appears that displays CodeWarrior preferences, as Figure 2.10 shows. We'll talk about this dialog box extensively in the next chapter.

For now, just close the Preference dialog by clicking the close box or the Cancel button. Then choose the Preferences item in the Edit menu. The same dialog box appears. The menu item and the toolbar icon do the same thing.

**6. Hide the toolbar.** Choose the Hide Toolbar item from the Tools menu; the toolbar disappears. Look in the Tools menu again. Notice that the same item now reads "Show Toolbar." Choose that item, and the toolbar reappears either anchored or unanchored, depending on what its state was when you hid it.

FIGURE 2.10

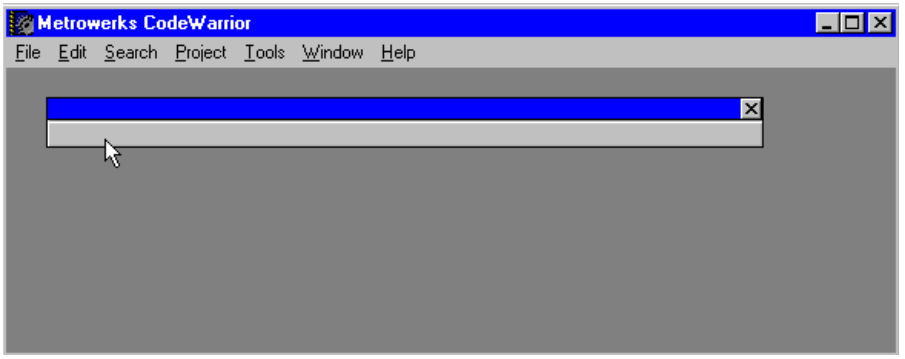
The Preference dialog box lets you set global options that affect all your CodeWarrior projects.




**7. Clear and reset the toolbar.** Choose the Clear Toolbar item from the Tools menu; all the icons in the toolbar disappear. This leaves only the gray message or status area, as shown in Figure 2.11.

FIGURE 2.11

The message area of the toolbar remains onscreen even when you clear all icons.




Now choose the Reset Toolbar item from the Tools menu, and all the icons in the toolbar reappear. The toolbar is once again fully functional as a one-click interface device.

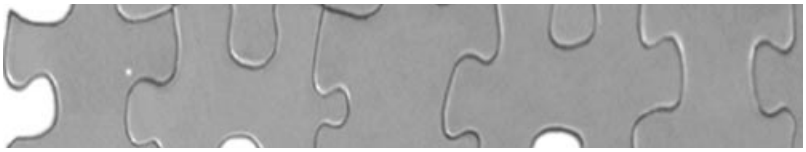
That's it for this exercise. You can exit CodeWarrior if you wish. However, feel free to experiment with the toolbar. Most of the commands won't do anything just yet, because you have just been introduced to the tools, and you haven't met all the functions that the toolbar icons represent. However, if you're feeling adventurous you should go exploring some more. This book covers all the important features of CodeWarrior systematically, but a self-directed preview can help you understand subsequent chapters.

Wherever you go exploring, use the menu, the toolbar, and the keyboard accelerators to perform the various operations. Decide which technique appeals to you, and the way that you prefer to work.

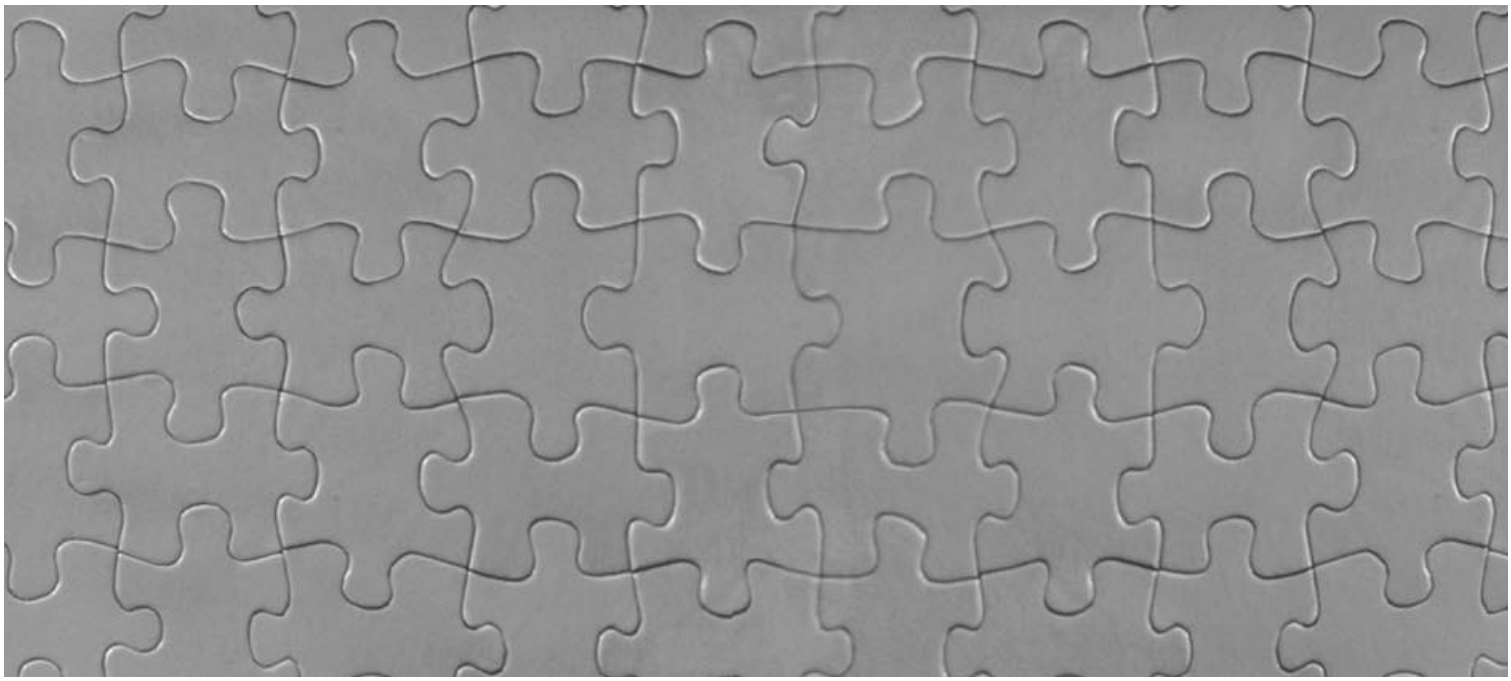
## Summary

While this chapter and exercise might have seemed simple, you have actually accomplished quite a bit. You have installed CodeWarrior, launched it, and started exploring. Congratulations.

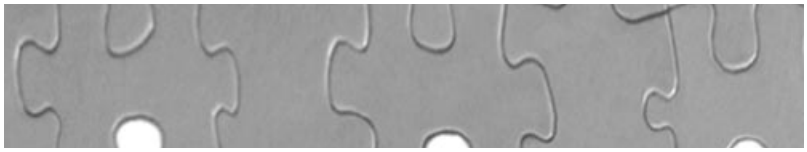
Now that you've had a brief glimpse of the basic IDE interface and what it looks like, it's time to really get down into the tools. Fundamentally, the IDE is designed so you can manage software development projects. And it is the project manager part of the IDE to which we turn first.



# PART



# II



# Managing Projects

Chapter 3 is the only chapter in this part of the book. In Chapter 3 you learn about the CodeWarrior project manager. Your project contains all the source code files required to build your software, and may contain many other related files such as precompiled libraries, related documentation and notes, and other essentials. The project manager is CodeWarrior's central engine, powering all the CodeWarrior tools. It's important enough to be the only chapter in this part!

After reading this chapter, you will know how to use the CodeWarrior project manager to create and manage a complete software development project.

# CHAPTER 3

THREE

# Working with Projects

- Understanding Projects
- File Management in the Project Manager
- Cool Features
- Setting Project Options
- Customizing Stationery
- Hands-On Exercises

This chapter covers the CodeWarrior project manager—your control center for managing files, compiling code, activating debugging, and much more. You will learn how to use the project manager to do all these things by the time you finish this chapter. Although you won't actually learn the details of tasks such as compiling or debugging until later in the book, in this chapter you'll discover how the project manager affects these tasks. Because projects are central to just about everything you do in CodeWarrior, this chapter gives you a real taste for the breadth of the tools.

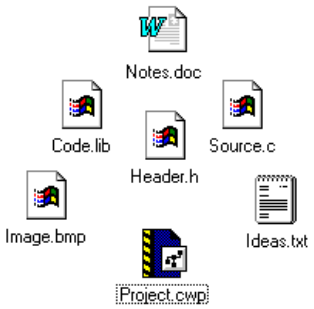
## Understanding Projects

A software development project is really a collection of many different files that you combine to create a finished program such as an executable application or library. You may have pictures, text, resource files, source code, interface files, libraries, sound files, video clips, documentation, design notes, object code, and so forth (see Figure 3.1).

A project is merely the concept that says that certain files are related. A project manager is the software that organizes all the different elements for you. A simple project might have only a few files, but a large project can have hundreds or even thousands of files.

A project manager does a lot more than just collect all the files that belong in a project. It also watches the files and keeps track of which ones have changed, which ones depend upon each other, and much more. You'll see what the CodeWarrior project manager does for you as we go through this chapter. For now, let's just stick with tracking which files belong in a project.

FIGURE 3.1
A finished program comes from many roots.




There are many different kinds of files, and some are more important than others. Some, such as your source code, are absolutely critical to the project. You can't compile your project without the source code.

## The Fundamentals of Compiling and Linking

When you write source code in a particular language, really what you're doing is following a set of syntax rules designed so that a compiler can read your code and understand what you want to do. The compiler reads your text, interprets it, and generates *object code* based on its interpretation. The object code is the set of machine-language instructions that make the computer execute the program you are creating.

Essentially, the compiler is a translator. It translates your source code from a high-level language such as C or Pascal into low-level machine code. It does this before you ever run your program. After the compiler is done, you can then run the same code over and over again. It never has to be translated again, and usually runs very quickly.

If your high-level code consists of more than one file (and it almost always does), the compiler generates a separate chunk of object code for each file. After the compiler is done, a *linker* puts all the pieces together into a finished package.

Other kinds of files are not so important. For example, you might have a file in which you keep notes about design decisions you make regarding a particular group of routines. This kind of information is important, but you can compile and build your final project without it. While it isn't essential that this file be in a project, it is nice to have handy access to this kind of information while you're programming. At the very least, it saves you the time required to hunt the file down when you want to look at it.

In addition to convenience, having access to all the files in one place gives you a subtle and more important advantage. If you have easy access to supporting documents and files, you are far more likely to use them. For example, your next bit of work might depend on a particular morsel of information, and you can't quite recall exactly what it is, or where it is. You know you've got it somewhere, but it's just too much of a bother to go hunting—so you make your best guess and move on. Unfortunately, you also know that on many occasions this tack sends you into troubled waters, leaving you in dire straits when it turns out your guess was wrong. Too late you realize you would have been much better off spending the time to find the file!

A good project manager tracks all your files for you, and keeps them at your fingertips. Because the supporting files are readily and easily available, you are far more likely to take the time to find the answers you seek. While it may not always be obvious, the improvement in your productivity as a result of easy access to supporting information is real—and significant.

A good project manager should enable you to add any kind of file to the project, and maintain easy and immediate access to it. Good software should enhance, not limit, your choices, while making it easier to accomplish your tasks. This chapter shows how the CodeWarrior project manager meets these goals. In the rest of this section, you'll learn how to create, open, and close a project in CodeWarrior. You'll also learn your way around the Project window.

## The Thought behind the Project Manager

Back in the "old days" of programming, before the advent of the IDE, advanced programming systems used makefiles. (If you are unfamiliar with the concept of a makefile, see "The Fundamentals of a Makefile" sidebar in Chapter 1.) While very powerful, the makefile process is fraught with tedium and subject to silly typographical mistakes.

The project manager is really a makefile with a graphical interface. You specify files that you want in the project, and which compiler to use for each. The project manager keeps track of the object code automatically. You also specify a linker, and the name of your final executable. It is the project manager that takes care of all the housekeeping details, and that's a real boon. The project manager is to a makefile as Windows is to DOS.

The Metrowerks engineers are hard at work improving the project manager. For example, when working on a major project you might have a single code base with several targets. You might want to compile code for Windows 95 and Mac OS. You might have a release version and a debug version of your project, each with different settings. Right now, you have to have separate projects for each.

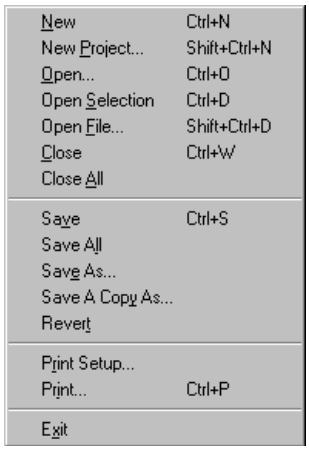
That's going to change. Eventually, the project manager will support multiple targets in the same project. All you will do is flip a switch, and the settings for the target you want will be put into play. This will make the project manager even more useful than it already is.

## Creating a Project

This is where it all begins. The first thing you want to do is create a new project, because you can't get much done in CodeWarrior without a project file. The command to create a new project is in the File menu, shown in Figure 3.2.

FIGURE 3.2

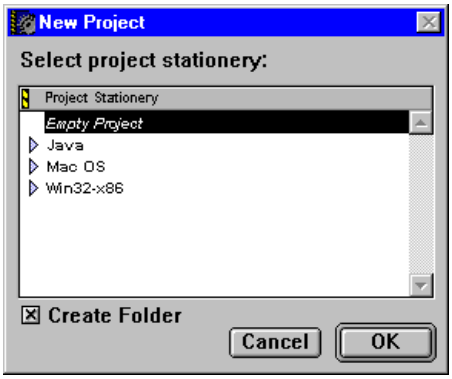
The File menu contains commands to create new files and projects.




When you choose the New Project command in the File menu (Shift+Ctrl+N), the New Project dialog box appears, as shown in Figure 3.3. Now things get really interesting.

FIGURE 3.3

Use the New Project dialog box to select stationery for a new project.




The New Project dialog box presents you with a choice. You can create an empty project and start from scratch, or you can create one based on an existing template project (also known as *stationery*) that has a lot of the preliminary setup work done. This dialog box provides

**NOTE**

The New item (Ctrl+N) creates a new text file, not a new project. You choose the New item when you want to create a new source code or interface file. (See the "Creating a New File" section in Chapter 4 on file management.)

you access to the templates so you can pick the one you want. The little triangles (called *disclosure triangles*) are analogous to the Windows tree controls, and as with those controls, you click on the triangle to disclose or hide the contents of a particular folder so you can navigate through the hierarchy to get to the template file you want.

**NOTE**

The disclosure triangle is a common feature of the CodeWarrior interface. You'll find it in every CodeWarrior tool. It always represents a container in a hierarchical list.

Depending on your version of CodeWarrior, you might see stationery for Windows 95/NT development, Java Virtual Machine, Mac OS, and perhaps other targets. Here the cross-platform power of CodeWarrior makes its first appearance. When you stop to think about it, the fact that such templates exist is astonishing! Fundamentally, programming for any target is really no more complicated than picking the right template. However, for the first part of this book I'm going to concentrate on the core tools. Later on, I'll visit each of the three main targets and talk about how to use this one environment to write code for each of them.

There may be several different templates for a given target. For example, inside the Win32x86 container is a collection of different kinds of stationery for Windows 95/NT development. If you click on the disclosure triangle, the item opens to reveal the stationery choices you have when programming for this target. Figure 3.4 shows you what the options might be. The actual items in this list will vary, depending upon the kinds of project stationery you have available, but whatever they are, stationery is a real boon. You can have stationery for all kinds of targets, and you can even create your

own. We'll talk about how to do that near the end of this chapter, after you have had an opportunity to learn more about how projects work and the kinds of options you have when working with projects.

## The Thought behind Stationery

On many occasions you must perform repetitive setup work. For example, a business letter requires some sort of return address information that never changes—the company the letter is from, surface and electronic mail addresses, phone numbers, and so forth.

Setting up a software project also requires some significant setup work. CodeWarrior is a complex piece of software, with hundreds of options you can set, dozens of paths to a single goal, and several major components.

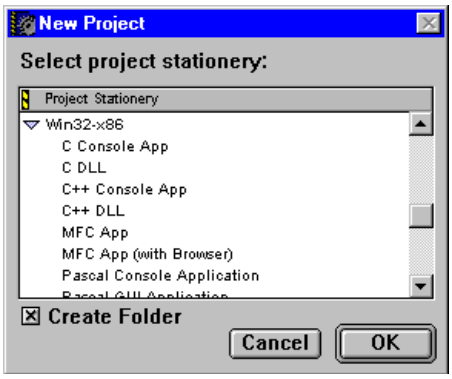
Of course CodeWarrior comes with default settings, but it's a safe bet that you're going to want to change some or many of those defaults. Setting up all these options while also adding a set of common libraries and files to your project just to get started is a chore, plain and simple.

The thought behind stationery is to eliminate this chore. To do away with part of the chore of producing a business letter, some bright person invented letterhead stationery. CodeWarrior stationery serves the same purpose. You prepare a common, baseline project with all the settings exactly as you like, and that includes all the libraries and common source files you want, and save it as stationery.

You then use the stationery as your starter package. Next time you want to create a new project, use the stationery as your letterhead. Then you don't have to do all the setup work. This saves you both time and aggravation, and frees you for productive work.

The available templates closely model the kinds of projects you might want to create. Select the one you want to use, and then click the

FIGURE 3.4
Choose the stationery that suits your project.




OK button (or press the Enter key) to accept that stationery as the basis for your new project.

**TIP**

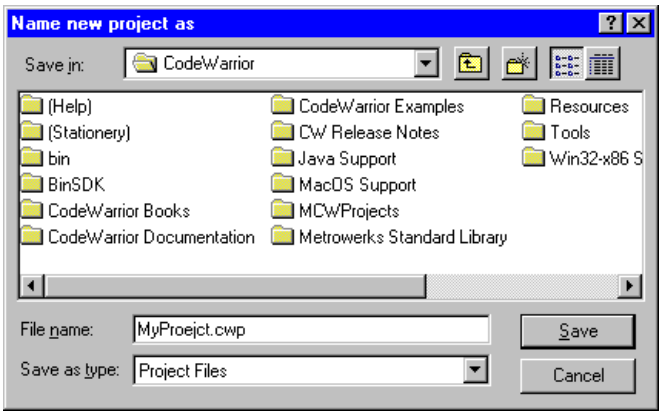
The Create Folder checkbox at the bottom left of the New Project dialog box is on by default. This tells CodeWarrior that you want a new folder to hold your new project. This is almost always what you want to do, because you'll have lots of files you want to keep together in the same directory. You can turn this option off if you want, although I wouldn't recommend it.

Now that you have specified the kind of stationery to use (or none, as the case may be), you have to specify the name for your new project, and where you want to put your new project folder. Another dialog box appears, as shown in Figure 3.5. This is the standard dialog box that enables you to put a file where you want it to be on your hard drive. You also enter the name of your project. You should end the project name with the extension **.cwp**, for CodeWarrior project. This lets you identify your project files easily, and enables Windows to launch the IDE when you double-click a project file on the Desktop.

When you are through setting up the name and location in the dialog box, click the Save button to create and save the new project. CodeWarrior makes a copy of the project template and all the neces-

FIGURE 3.5

The standard file dialog box lets you change directories and save a file.




**TIP**

Actually, if you forget to add the **.cwp** extension, CodeWarrior does it for you automatically if you have the Save as Type option set to save a project file.

sary files, and puts them in your new project folder (assuming you left the Create Folder check box turned on). (The new folder has the same name as your project, but without the **.cwp** extension.) Then a new project manager window appears.

Let's take a look at the Project window in detail.

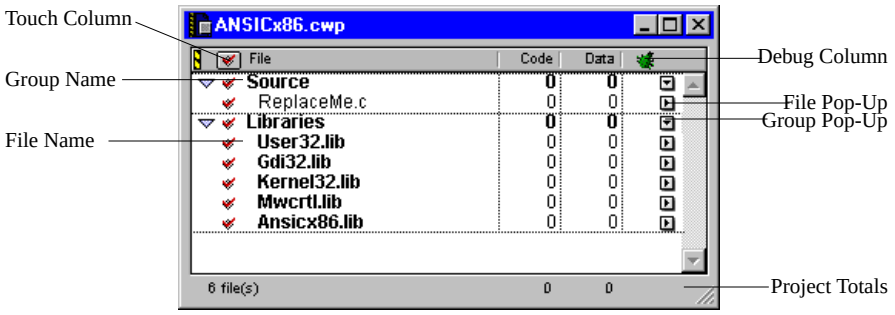
## The Project Window

Unless you build it with stationery, a new Project window is empty, with no files. Figure 3.6 shows you what a Project window looks like after you have added files to it. (I'll talk about how to add files a little later.) The window is really a list of all the files in the project, organized into logical groups.

Most of this window is informational. In that spirit, most of this discussion is background information. How you actually use the features of the Project window becomes clear later on in this chapter.

FIGURE 3.6

The Project window lists every source file in the project.




Each group name is the title of a hierarchical list, and you'll see the familiar disclosure triangle to the left of the name. You can open or close any group as you wish—just click the disclosure triangle to toggle the state of that group.

Each file in the project is listed in the File column. Files that appear in plain text can be opened, such as the **ReplaceMe.c** file, which is a source code file. You can open such a file because you have the actual text available for review. Files that appear in bold print cannot be opened by the project manager. For example, in Figure 3.6 the **Ansicx86.lib** file cannot be opened. In this case, this file contains the Metrowerks standard C library, which has been included in this project as a compiled library, not as source code. As a result, no source code file exists, only the actual object code that makes up the compiled library.

The check mark to the left of a file name indicates that the file needs to be compiled. A file needs to be compiled either because it has never been compiled, or you have changed the file since the last time it was compiled.

**TIP**

A check mark next to a group name tells you that one or more files in that group need to be compiled. If no check mark appears next to a group name, all of that group's files have been compiled. This lets you see the status of a group, even if the group is closed and the individual files are not displayed.

## The Fundamentals of Code Libraries

Every programmer wants to reuse code, because it saves a lot of work. After all, why reinvent the wheel? A code library is one form of reusable code.

A code library contains common or generic routines that you might want to use in a variety of programs. The standard C library that is part of the C language is a good example. It includes routines to manage memory, manipulate strings, handle simple input and output to a character-based console, perform trigonometry, and so forth. When you include it in your project, your code can call the library routines. This saves you having to write a lot of code to perform common tasks. CodeWarrior comes with standard libraries for C, C++, and Pascal.

You put a code library into a project in one of two ways. If you have the source code, you can include the code just as you would your own. As far as the project is concerned, the library is the same as any other source file. This approach takes more time, because the compiler has to work on all the library code as well as your own.

You can also include a library as a precompiled chunk of object code. You create a separate project that has all the library source files in it, and then compile that code as a code library. Once you have the compiled library, you can add it to any other project. Such a library is called a *static-link* library, because you include it directly in your project. When you build your final application, the library's object code is copied directly into it, and becomes an integral part of the executable.

continued ➠

## The Fundamentals of Code Libraries

Of course, you have to be careful that the library was built using the same compiler settings as your project, or inconsistencies in how your code works with the library can occur. This is not a case where you can mix chocolate and vanilla. All the code in a project has to be compiled as the same flavor. When you include a library as source code and compile it as part of the final project, this is not a problem.

Another way to build a library is called a *dynamically linked library* (DLL). A DLL is really a code fragment, much like a static-linked library. However, your executable connects to the DLL at runtime (that is, it links dynamically), so you don't include the DLL as part of the project. When your executable runs, it must be able to find the required DLL, to link to it. Once it does, your executable can then call routines in the DLL.

The beauty of a DLL is that more than one executable can use the same DLL at the same time. This means that if you have different modules that need to share the same routines, you can put those routines in a DLL, thus saving space in memory and on disk. You reuse the common code. Another advantage is that when you update the DLL, all executables that use it receive the benefit of the change. For example, let's say you had a DLL that implemented a data compression algorithm. You have three different applications that use this DLL. One day, you discover a way to get better compression faster. You rewrite the DLL, replace it on your disk, and all three applications are suddenly faster and more efficient. Cool!

You have to be careful when you update DLLs, because the names and calling parameters for the routines in the DLL have to stay the same. Nevertheless, DLLs are an excellent way to reuse code.

The Code and Data columns display the amount of object code and global data created by the compiler when you compile the file. For a group, the totals for that group are displayed. Along the bottom of the window, the complete project total appears, including the number of files and the total amount of code and data.

To the right of the Data column is the Debug column. A mark in this column means that debugging information will be generated for that file.

**NOTE**

The project total is not necessarily the same as the size of your final executable. The CodeWarrior linker removes any unused code when it makes the executable, so your final executable could be smaller than the generated code and data.

**TIP**

In addition to the check mark, you can have a debugging mark next to a group name as well. A mark next to a group name means one or more files in that group are set for debugging.

Finally, at the far right of the Project window, each file has a pop-up menu, indicated by the little triangle button. Click on the button, and a pop-up menu appears that lists a variety of options depending on the nature of the file.

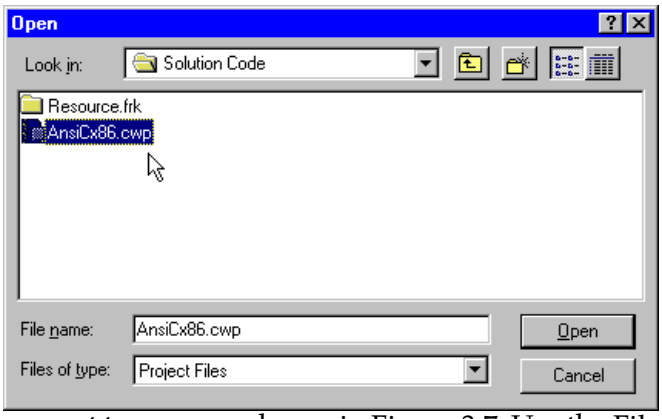
You have met the Project window, and you can see how it lists all the files in a project, and lets you control compiling and debugging. You know you can add files and libraries, and there have even been hints about such things as compiler settings. You're going to learn a lot more about all of these things, but right now, lets continue to discuss working with projects as a whole.

## Opening a Project

You'll often want to open an existing project, rather than create a new project—every time you want to resume work on a project you have previously closed, for example. Of course, you can use any standard technique for opening a file from Windows 95/NT. Within CodeWarrior, you simply choose the Open command in the File menu (Ctrl+O). When you do, a dialog box appears that asks you to locate

FIGURE 3.7

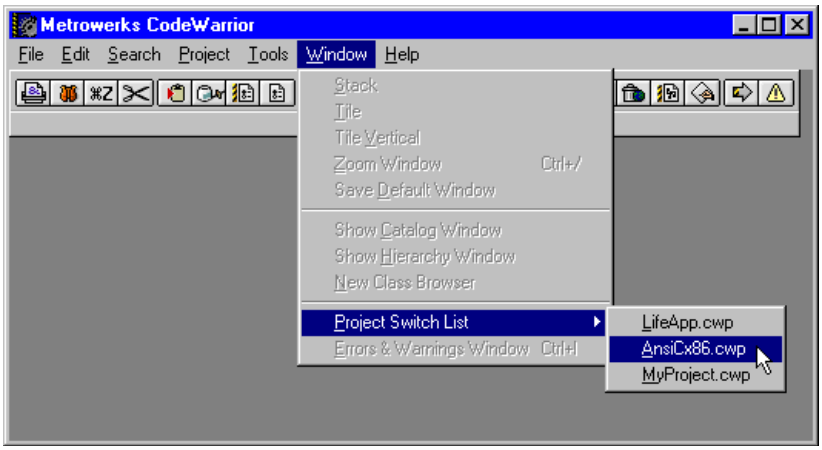
The Open dialog box lets you open either project files or source files.




the file you want to open, as shown in Figure 3.7. Use the Files of Type combo box to specify that you want to open a project file, choosing from files ending with the **.cwp** extension. When you select the project file you're interested in, click the Open button.

FIGURE 3.8

Use the Project Switch List to open a project quickly and easily.



**TIP**

Here's a fast way to reopen any project you have already had open during your current session with CodeWarrior. In the Window menu, select the Project Switch List item and a submenu appears listing every project you have had open. Simply select the project you want to open. Figure 3.8 shows the menu.

## Saving a Project

There is no need for you to ever explicitly and intentionally save a project. CodeWarrior saves projects automatically. You do need to save individual files in the project, but that will be covered in the next chapter.

You can make a copy of a project file, however. For example, you may want to share a project with a colleague, back up your work, or create stationery. This is an easy task to accomplish. With the project file as the active window, choose the Save a Copy As command from the File menu. In the standard dialog box that appears you can save a copy of the project anywhere you want.

**TIP**

When you save a copy of a project, you can save it as plain text if you want. When you do, it is no longer a project file. You get a list of every group and file in the project. This can be very useful for archival or documentation purposes.

**WARNING**

Saving a copy of a project file does not save the contents of files contained in the project. This is simply a way to make a copy of the existing project. You'll learn about saving individual files in the next chapter.

## Closing a Project

Closing a project is simple. You can click the Close box of the Project window, just as you can for any window. Alternatively, when the

Project window is the active window, choose the Close command in the File menu. CodeWarrior closes the project. Any changes that have occurred to the project (such as changes in the file list from adding or removing files) are saved automatically.

At this point, you have a good feel for the Project window and what it represents. You can create, open, close, and save a project as a whole. And you know all about stationery. The next critical feature you need to know about projects is how to manage files in a project.

# File Management in the Project Manager

The true purpose of the project manager is file management. You know that a project can require dozens, if not hundreds or even thousands, of files. The project manager keeps track of those files. You need to know how to add and organize files in the project, and how to work with the files using the project manager. That's what this section is all about.

## Selecting Files

As you undoubtedly know, the process of working with one or more files is simple. First, you select the file or files you want to work with. Then you issue the command that does something with these files. CodeWarrior uses this same design—first you select, then you act—just about everywhere.

In this section, you learn how to select a file in the CodeWarrior Project window. It's really quite simple, and exactly what you would expect from a good, friendly program. You can select one file, or any arbitrary set of files.

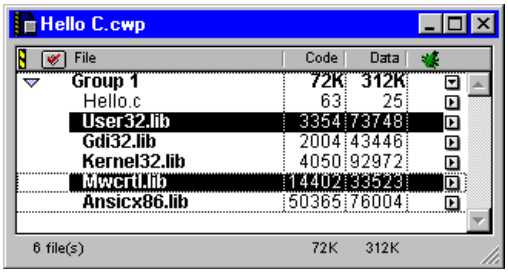
To select a single file, simply click the name in the Project window. You can also change the selection using the arrow keys. Use the Up or

Down arrow keys to select the previous or next file in the list. To select a range of files, first select a file at one end of the range of files; then press the Shift key and click the file at the other end of the range. All files in the range, including the files at both ends, become selected.

To select discontiguous files, press the Ctrl key and click a file. Any previously selected file remains selected, and the file you click is added to the selection. Figure 3.9 shows you what this looks like.

FIGURE 3.9

The Project window supports discontiguous selection of files so you can work on exactly what you want.




If you Ctrl+click a file that is already selected, it becomes deselected.

In addition to clicking and arrow-key navigation, you can also type the name of a file. As you type, the letters appear in the message area of the toolbar, and the first file beginning with those letters is selected. The Metrowerks documentation calls this feature *type-ahead selection* and it is used in many places throughout the tools. Most lists you encounter in CodeWarrior, such as in the code browser, the debugger file pane, and many other places support type-ahead selection.

Using these techniques, you can select any individual file or any arbitrary group of files you want. Once you select a file, then you can do something with it. The rest of this section discusses some of the things you can do with files in the Project window, such as adding a file, removing a file, moving a file, and so forth.

## Adding Files

As you create new source files, you must add them to the project. Two issues arise with respect to adding files. First, what kinds of files can you add to a CodeWarrior project? Second, how do you add a file?

### What Kinds of Files Can You Add?

You can add any kind of file that a particular CodeWarrior project recognizes. Because you can tell CodeWarrior what files to recognize for each individual project, you have absolute control over what kinds of files can be added.

CodeWarrior automatically recognizes any common file you are likely to want to add to a project, including source files, libraries, precompiled headers, and resource files. It recognizes files based on the naming extension associated with each file type. You can modify the list of recognized file types, but the way you do this is wrapped up in picking a target and associating files with compilers. Discussing those topics right now would take us far afield. So, I'll continue to concentrate on the subject of managing files. However, before you are through with this chapter I'll explain how you can add a new file type to the list of legal files, and how to associate each file type with a particular compiler.

### How Do You Add a File?

The CodeWarrior project manager has several techniques you can use to add files to a project. You can add a single file, a whole folder of files, or an arbitrary set of files.

The simplest way to add a single source file to a project is to have that source file open. When that file is the active window, choose the Add Window command from the Project menu. When you do, Code-

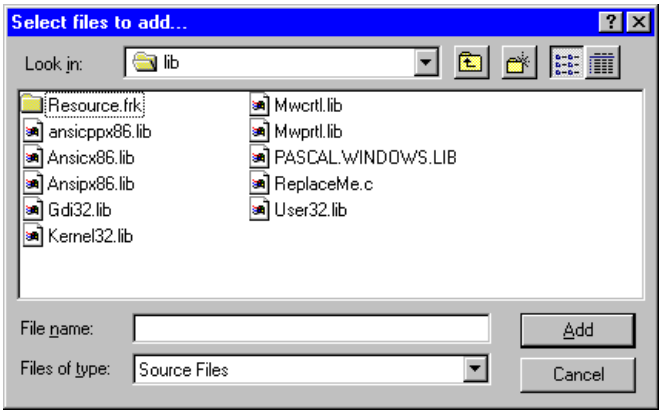
Warrior adds the file associated with that window to the project. If the file hasn't been named yet, you must save and name the file.

**TIP**

Newly added files appear after the currently selected item in the Project window. If no item is selected, the new file appears at the end of the project.

To add one file or a whole group of files, you can also choose the Add Files command from the Project menu. When you do, the Add Files dialog box appears, as shown in Figure 3.10. It is the standard Windows Open File dialog box.

FIGURE 3.10
Use the Add Files dialog box to add one or more files to a project.




You can see source files or resource files using the Files of Type combo box. Use the standard dialog box controls to move to the folder that contains the files you want to add. Then use the mouse to select the file or files you want. As you do, the names of the files appear in the File Name section of the dialog box. You can select a range of files with the Shift key, and select discontiguous files using the Ctrl key. As you do, the names of all selected files appear in the File Name box.

When you are done selecting files, click the Add button. CodeWarrior adds the files to your project. If you want to add files from another folder, simply repeat the process.

## Creating and Naming Groups

When a project grows to more than just a few files, you will want to organize the files into groups of related files. This can make navigating around the Project window a lot simpler. For example, you can keep libraries together in one section of the Project window, so you can always find the libraries easily. You can keep different categories of source files in different groups, such as all files related to a particular window, or all notes and documentation.

In the Project window, a group name is bold and has a disclosure triangle to the left of it. That triangle is diagnostic—if it has a disclosure triangle, it is a group name. You can display or hide the files in a group by clicking this triangle. If you Alt+click the triangle, all groups open or close.

**TIP**

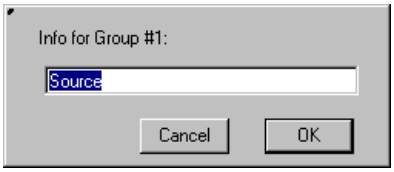
When a group name is selected, the Left Arrow key closes the group, and the Right Arrow key opens the group.

You can create a new group in two ways: splitting and dragging. If you want to split an existing group into two, select the file that you want to be the first file in the new group, and then press Ctrl+Enter. The project manager creates a new group at that location, and includes every file from the selected file down to the end of the original group.

An alternative exists: You can simply select a file or files from any location in the project manager window (they can even be in different groups), drag the selected files to the bottom of the Project window, and drop them in the empty white space below the last group. CodeWarrior creates a new group and places the files in that group.

You can also change the name of a group by selecting a name and then pressing Enter, or double-clicking the name. When you do, a dialog box appears, as illustrated in Figure 3.11. Simply enter the name you want to give to the group.

FIGURE 3.11
You can set the name for any group.




## Moving Files and Groups

You can reorganize files within, or move files between, groups, or change the order of groups within a project. Simply select the items you want to move, drag them to a new location, and drop them there. It's that easy.

As you drag, the Project window displays a gray focus bar to indicate the position that the selected items will assume in the list when you drop them. Just move the mouse until the focus bar is where you want it.

For example, Figure 3.12 illustrates the process of relocating a group in the Project window. To move a group, first select the group's name. Then drag it to the desired position. The entire group moves as one.

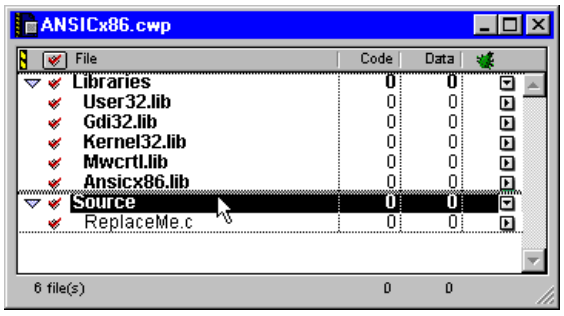
To move a file or files to a different group, simply drag the selected file(s) to the new location. This process works even for a discontiguous selection from multiple groups.

## Removing Files and Groups

Removing one or more files, or one or more groups from a project is extremely simple. Select the files and/or groups you want to remove

FIGURE 3.12

Drag a group name to reorganize the order of groups in the project.




from the project. Then choose the Remove Files command from the Project menu or press the Delete key. When you do, CodeWarrior removes the files from the project. This command works for both files and groups.

If you remove a group, you remove all the files in that group.

**NOTE**

Removing a file from a project does not delete the file from your hard drive. It simply removes the file from the project. You can always add the file back into the project later on.

Table 3.1 summarizes what you can do with groups, and how to do it.

## Opening Files

The project manager is best at giving you an easy way to access and open files. Because this is such a critical feature of the manager, it gives you several ways to open a file. You can use whatever technique appeals to you at a given moment.

TABLE 3.1 Working with Groups in the Project Window

| Goal | Action |
|---|---|
| Create | Select a file and press Ctrl+Enter; or select file(s) and drag to bottom of project manager window |
| Rename | Select a name and press Enter; or double-click a name |
| Move | Select a group and drag to new location |
| Remove | Select a group and choose Remove Files from Project menu; or select a group and press Delete |

Perhaps the most direct technique is to double-click a source file name in the Project window. When you do, CodeWarrior opens the associated file and displays it in an editor window. If the file is not a source file, CodeWarrior attempts to open it using the application that created it. You can also select multiple files, and press the Enter key. When you do, CodeWarrior opens all of them.

Use the Open command from the File menu to open any recognized file, even those not in the project. When you choose this command, the standard dialog box appears allowing you to locate and open a file (see Figure 3.7).

You have still another way to access project files. To the right of each name, at the far right of the Project window, is a little icon representing a pop-up menu. The pop-up menu for a group, illustrated in Figure 3.13, lists every file in the group. You can open the file by selecting it from this menu. This is particularly useful if the group is closed.

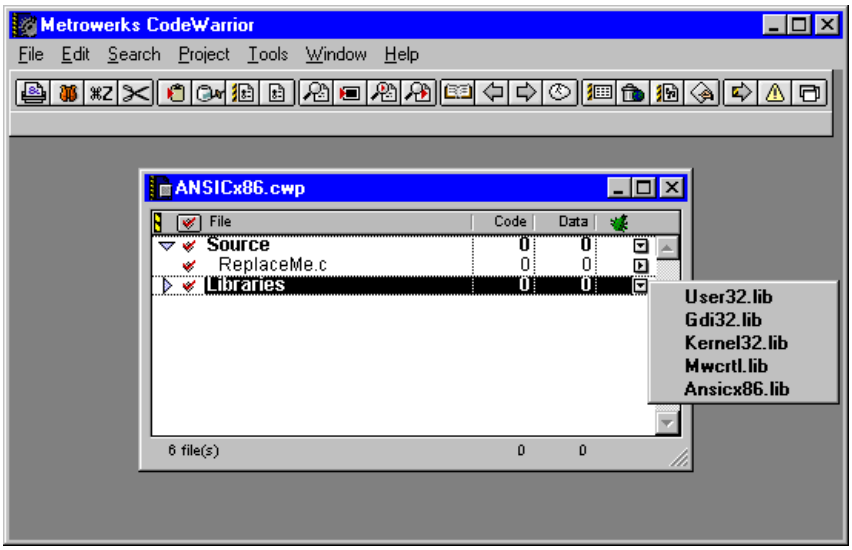
As discussed later, the pop-up menu for an individual file is a little different.

## Accessing Included Files

Many files directly connected to a project don't normally appear in the Project window. These are the interface files included in the

FIGURE 3.13

The group pop-up menu lists all files in the group.

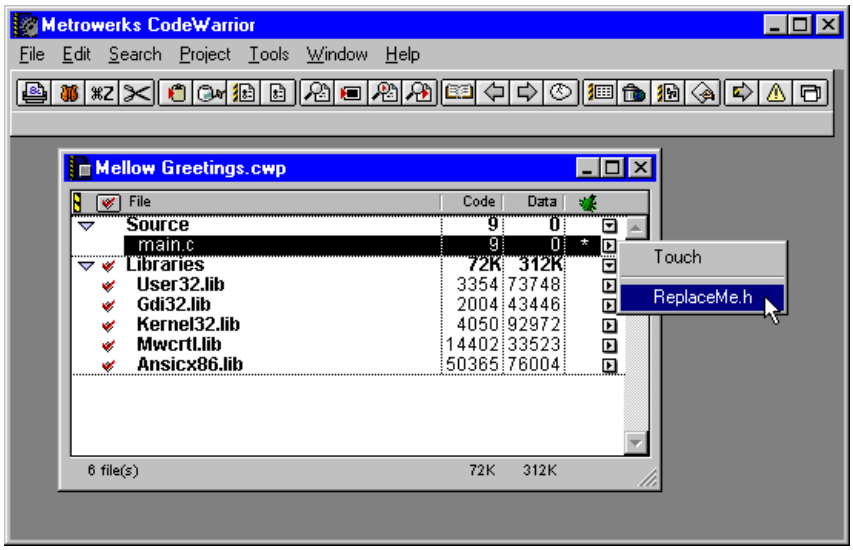



source files, such as C or C++ header files. Wouldn't it be nice to have a list of every interface file included in a source file, and be able to open that file quickly and easily? That's what the file pop-up menu is for. Figure 3.14 shows the menu.

This menu has a Touch or Untouch command (we'll talk about those in a minute), and then lists every interface file included in the source file. To open an interface file, simply select the item from the menu. CodeWarrior opens it in an editor window. I'll discuss the editor in the next chapter.

FIGURE 3.14

The file pop-up menu lets you touch a file, and lists all included files.




The CodeWarrior project manager is very good at giving you quick and easy access to your files. This same ease of use becomes evident in other aspects of CodeWarrior.

**NOTE**

The list of included files in the file pop-up menu only appears after a file has been compiled for the first time. The compiler must read the file to see what interface files are included.

For now, you can learn about a few other things the Project window can do.

## Cool Features

The project manager gives you precise control both over which files are compiled and over which have debug information.

## Touching Files

A check mark to the left of a file name tells you that it is scheduled to be compiled. CodeWarrior usually takes care of this for you automatically. It tracks the modification date and time of every file in the project. The project manager compares that date and time against the last time it was compiled. If the dates do not match, the file is marked for compiling.

However, the project manager lets you override this automatic tracking. You can turn compiling on or off for any file at any time. In CodeWarrior terminology, *touching* a file means that you want the compiler to update the code the next time you build or update your project.

So, if you *touch* an untouched file, you tell the project manager that you want to recompile it. If you *untouch* a touched file, you are telling the project manager that you do not want to compile the file. To touch a file, you can just click the Touch column to the left of the file name. You can also choose the Touch command from the file pop-up menu that appears at the far right of the Project window beside the file name. If it has already been touched, the item reads Untouch so you can untouch the file.

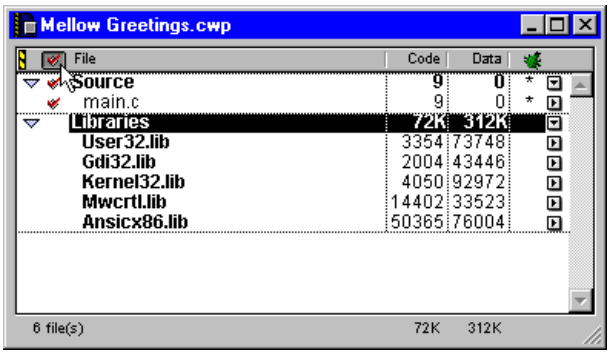
CodeWarrior also lets you work with third-party source code editors. However, when you modify a file in some other editor, CodeWarrior probably won't know that the file has been changed. You can force CodeWarrior to check for file modification date and time, and compare it against the compiled date and time. This is called *synchronizing modification dates*.

You can choose the Synchronize Modification Dates command from the Project menu, or you can simply click the corresponding check mark button at the top of the check mark column, as shown in Figure 3.15.

The project manager gives you absolute and complete control over which files are compiled. Most of the time, however, you'll just let CodeWarrior track everything for you automatically.

FIGURE 3.15

Clicking the Touch Column button tells CodeWarrior to check that all files are current.




## Controlling Debugging

As you know, the debugger is a separate application in CodeWarrior. However, the debugger needs symbol information to debug a file, so it can track the relationship of symbols in your source code to the corresponding object code created by the compiler. This information is created on a per-file basis by the compilers and linkers, under the control of the project manager. You can use the project manager to generate this information, or not.

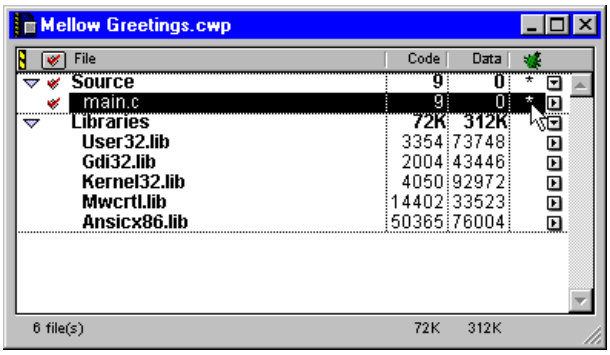
To generate debugging information for a file or group, simply click the Debug column in the Project window, to the right of the file name. When you do, a mark appears telling you that this file will have debug information generated. Figure 3.16 shows how it works.

If you have debug information, when you run with the debugger later on, it will be able to display the source code for this file. (The debugger is discussed in detail in Chapter 8, “Debugging Your Code.”)

You can change the debug state of an entire group of files if you click the Debug column next to the group name.

FIGURE 3.16

Click the Debug column for a file or group to control the creation of symbol information for the debugger.




**TIP**

You can change the debug state of every file in the project if you Alt+click anywhere in the Debug column. (A file that does not have source code, such as a precompiled library, cannot be debugged.)

That about wraps it up for what you can do with the Project window. It's quite a list of features. You can add, relocate, remove, and open files and groups. You can create and name groups at will. You can also control which files are compiled, which ones are debugged, and even have immediate access to included files that don't appear in the project list.

However, that's not everything you have to know with respect to projects. As I have mentioned before, every project has preferences and settings associated with it.

# Setting Project Options

When you create and work with a project, you have many options to control. For example, you decide what language you want to use, what target you are programming for, how you want the compiler to optimize your code, what kinds of compiler warnings you want to see, and so forth.

In CodeWarrior terminology, you have two kinds of options: *preferences* and *settings*. Preferences are options that apply to CodeWarrior globally, and are the same for all projects. Settings are options that apply to a project individually, and may vary from project to project.

Preferences and settings affect every facet of CodeWarrior programming. As a result, you will encounter them throughout this book. For example, when I talk about the source code editor, I'll also discuss the preferences and settings for the editor.

In this section I'll first explain the interface for CodeWarrior options. Then I'll discuss how to control some project-specific settings. Subsequent chapters will cover options for compilers, linkers, the editor, the browser, and many other aspects of CodeWarrior tools.

## Understanding CodeWarrior Options

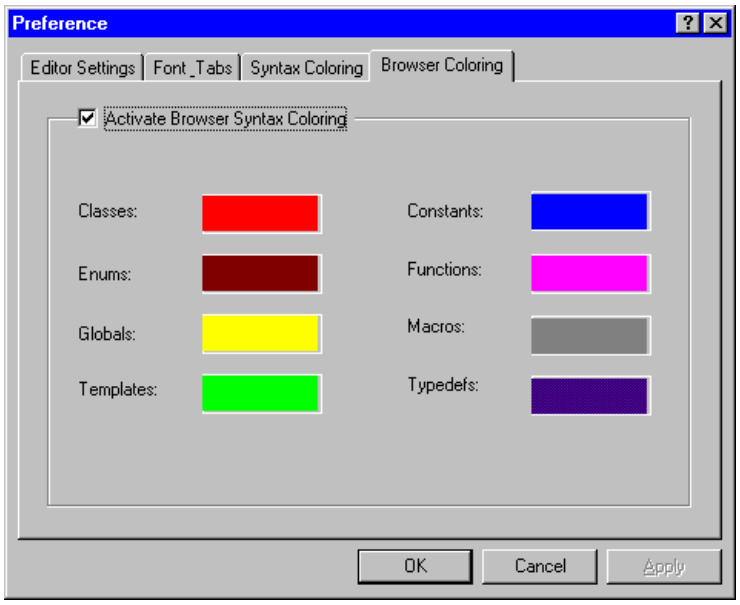
As I mentioned previously, options consist of both preferences and settings. The interface for both kinds of options is the same. I'll use the Preference dialog box as the example, but the discussion applies equally to the Settings dialog box.

To open the Preference dialog box, choose the Preference command from the Edit menu. (Remember, you can click an icon in the toolbar as well.) The Preference dialog box will appear as shown in Figure 3.17.

Groups of related options are organized by topic into a single panel. The available panels appear as tabs along the top of the dialog box.

FIGURE 3.17

Use the Preference dialog box to set global preferences.




Click a tab to display the panel. For example, Figure 3.17 shows the preferences for the colors that the browser uses to display various kinds of symbols.

The buttons along the bottom of the dialog box let you accept all options in all panels, or cancel any changes you have made in any panel.

**WARNING**

If you click OK or Cancel, the dialog box closes. Changes to options are saved and take effect only when you click the OK button. You can switch panels and change options in several panels without closing the dialog box. However, if you click Cancel for the last panel, *all* changes you have made to *every* panel since you opened the dialog box are lost forever in a sea of electrons.

CodeWarrior automatically saves all preferences when you quit the application. When you relaunch CodeWarrior, the preferences remain exactly as they were when you last quit.

The project manager handles project settings in exactly the same way. When you close a project, all settings are saved automatically. When you reopen a project, all settings are restored to what they were when you closed the project.

## Setting Default Project Options

When you create a new project, CodeWarrior provides default values for every option in every preference and settings panel. Earlier we discussed how you could set up project stationery to create default projects with particular settings. You can also modify the automatic defaults that CodeWarrior uses for a new empty project, that is, a project created with the New Project item in the File menu.

To modify CodeWarrior defaults, first make sure that *no* project is opened. Then, open up the preferences and settings panels and set the values you would like. Because no project is opened, the options you choose will apply to all new projects. (You'll encounter each panel and the options on each panel in the course of this book.) Whenever I discuss a particular setting, just remember that if you set the value with no project open, that setting becomes the default value used for all new projects.

**NOTE**

Remember, if you use stationery as the basis for a new project, the settings in the stationery project override the global defaults you set when no project is open. The automatic defaults only apply to a new, empty project.

What values should you choose for all these options? Well, that's what this whole book is about—how to use CodeWarrior (and its many options) to write code. In fact, you can start learning the answer

to that question right away. Now that you have seen how the options interface works, you can look at some of the settings that relate to an individual project.

## Working with Settings

Any settings panel is project-specific but some are directly related to how the project manager itself works. Those are the panels I'm going to discuss right now; they are as follows:

- Target
- x86 Settings
- Build Extras
- Access Paths

To see any of these settings panels, first choose the Project Settings command from the Edit menu, or click the appropriate icon in the toolbar. As with Preferences dialog box, select the tab for the panel of interest to you to see the options.

## Picking a Target

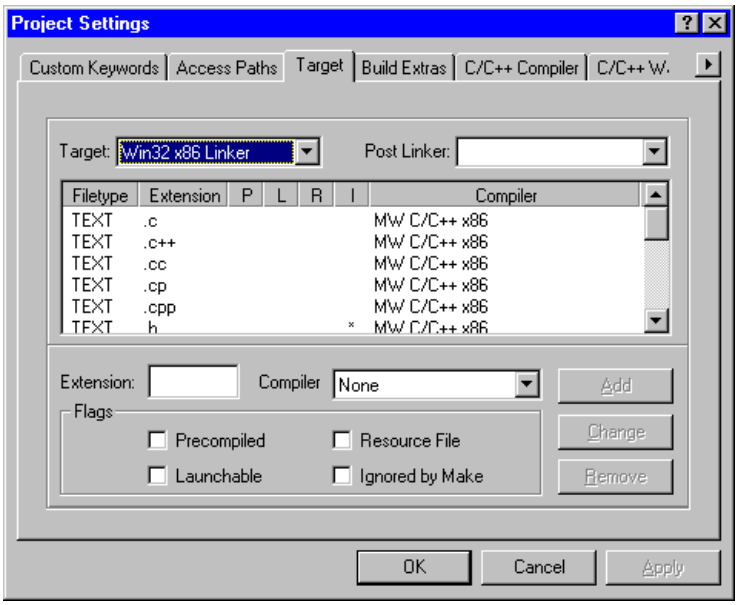
One of the first things you have to do when you create a new CodeWarrior project is to pick a target. If you used project stationery, the target is probably already set for you. However, it's worthwhile to take a look at the target settings because in this panel you can do more than pick a target.

When you open the Target settings panel, you'll see a panel like Figure 3.18.

The first thing you do is pick the target you want from the pop-up menu at the top left of the panel. The menu lists all available targets

FIGURE 3.18

Use the Target settings panel to specify the target for which you are writing code.




for your version of CodeWarrior. After you set the target, you can turn your attention to the rest of the panel.

**NOTE**

When you pick the target, what you are really doing is choosing a back-end compiler and linker. We discussed the CodeWarrior compiler architecture in Chapter 1. At that time I mentioned that I'd remind you when the IR—the intermediate representation—made a subtle appearance. Here is one case where you can see IR's effects.

**TIP**

In this and all other panels, you can move from item to item by pressing the Tab key, which changes the focus from one item in the dialog box to another.

In Figure 3.18 you also see a combo box you can use to set a *postlinker* (which performs work on the executable after it has been linked). Most targets do not require a postlinker, but CodeWarrior supports a postlink phase in the compiling/linking process.

The rest of this panel is devoted to setting up which file types use which front-end compiler. A scrolling list shows the types of files that CodeWarrior recognizes for this target. The default list will probably take care of most of your needs. However, you can add, remove, or modify the contents of the list using the buttons below the scrolling list.

You specify a naming extension, pick the compiler to use for that kind of file, and specify some flags. The flags tell CodeWarrior whether this kind of file should be precompiled, is launchable, is a resource file, and should be ignored by a Make command (that is, ignored by the linker).

When you have new data entered in the bottom part of this panel, the Add and Change buttons become active. The Add button adds a new entry to the list, and the Change button changes the currently selected entry. The Remove button removes the currently selected entry from the list.

**TIP**

The project manager allows you to add any file to the Project window whose extension is listed in this panel. In the default list, for example, you'll see entries for files ending with **.cp** and **.cpp**, because some programmers prefer one convention over the other for naming C++ source files. Both are listed so that you can add files using either naming style.

## Choosing Project Settings

The Settings dialog box has a project settings panel for each kind of target. For example, project settings panels are available for Mac OS

PowerPC, Mac OS 68K, Java, and x86 projects. Figure 3.19 shows the x86 Project Settings panel.

The precise contents of the settings panel vary for each target. I'll discuss the individual panels in detail when talking about programming for each of these targets. However, you probably won't need to deal with this panel very often. If you use project stationery, this panel is already set to default values.

**NOTE**

An interesting thing happens when you choose a target. The list of available panels in the Settings dialog box changes. Certain panels are appropriate only for certain targets. CodeWarrior helps you by limiting the panels to those that could have some impact on your project. For example, it doesn't make a lot of sense to set the Mac OS 68K options if you are building Windows 95/NT code.

## .Setting Build Extras

The Build Extras settings panel has a few more tidbits that you can set for your project. Figure 3.20 shows the options available in this panel.

CodeWarrior compares file modification date and time against its own records to determine which files need to be updated. The Use Modification Date Caching checkbox tells CodeWarrior you want to cache its date information in memory. If you turn this option on, CodeWarrior will be a little faster checking whether files have changed. However, it also means that CodeWarrior does not go to disk every time, and so the data in the cache could become outdated. This might happen if you use a text editor other than the one in CodeWarrior to modify a source code file for example.

This panel is also where you turn on the source code browser. The browser is discussed in detail in Chapter 6.

The other two check boxes affect Pascal code only. If you turn on the Generate Make Map File option, the Pascal linker generates a special text file that provides you with a list of dependencies, and the compilation order. If you turn on the Store Analysis Results option, the com-

FIGURE 3.19

The x86 Project Settings panel controls how you work with a Windows project.

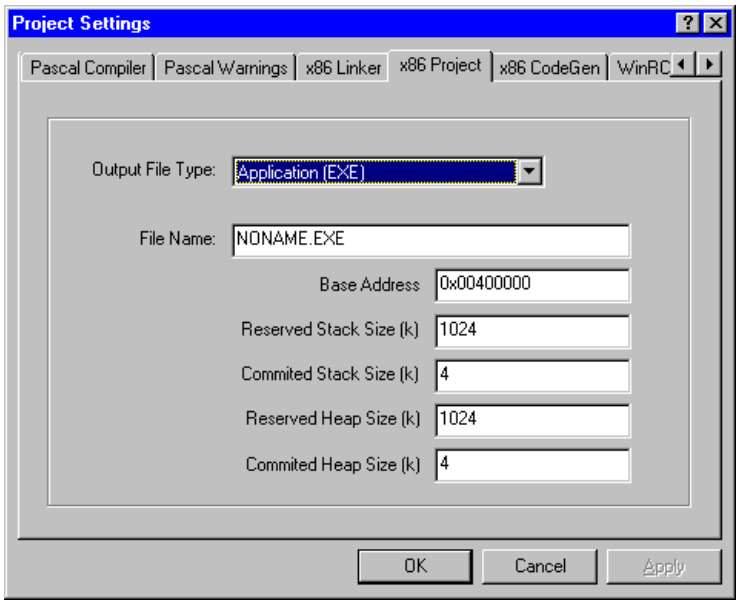
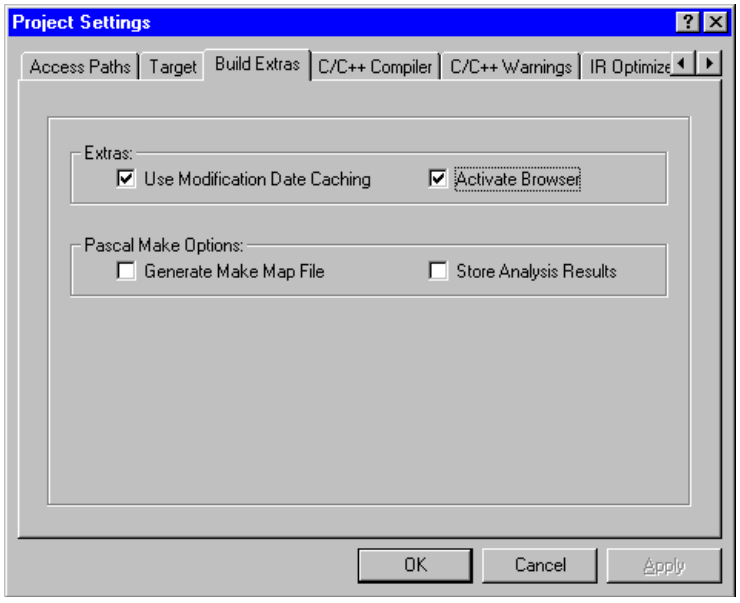



FIGURE 3.20

The Build Extras settings panel controls the browser and other miscellaneous features.



piler preserves Pascal dependency information between compiles. If this option is on, compiling a project takes longer the first time, but subsequent builds will be faster.

## Setting Access Paths

The last panel that affects your project directly is the Access Paths panel. In CodeWarrior terminology, an *access path* is a directory where CodeWarrior looks for files when building your project or hunting for files.

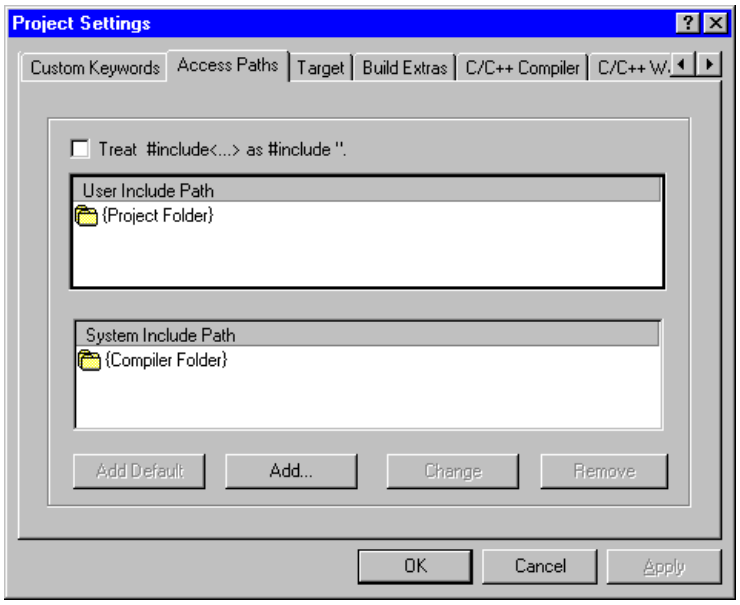
CodeWarrior has two sets of folders in which it looks for files: the User Include Path and the System Include Path. The User set typically contains the source files particular to your project. The System set typically contains libraries, headers, and the other files you would use over and over again in different projects. By default, the search looks in the specified folder, and inside *every* subfolder within that folder.

When CodeWarrior looks for a file, it goes through each of the folders listed in the Access Paths panel (and their subfolders), starting at the top and working down the list. The User set is searched first, followed by the System set. Within each set you can control the order of folders by simply dragging and dropping a folder to a new location in the list. You can even drag a folder from one list to the other. Figure 3.21 shows this panel. Use this panel to add, change, or remove a folder from either the User Include Path or the System Include Path.

One particular convention that should be noted is the folder in braces, the {Project Folder} in the User set. This path does not represent an actual folder with that name. It is the folder that contains your project file. Because CodeWarrior looks inside every folder inside the specified folder, this means that if you keep your source code inside the same folder as your project file (either at the same level or in subfolders) CodeWarrior will find your source files without fail.

FIGURE 3.21

The Access Paths settings panel tells CodeWarrior where to look for files.




**TIP**

You can force CodeWarrior to ignore the contents of any directory by naming the folder within parentheses. For example, a search will not look inside a folder named **(Do Not Look Here)**.

CodeWarrior tries to make working with access paths as easy as possible. While you can come here and manually modify the folders in which CodeWarrior looks for files, you have an easier way to add a new access path. Just add a file to the Project window. If the path to that file is not already in the access paths for the project, CodeWarrior adds that access path automatically (and tells you it has done so).

You know that the search normally starts with the User Include Path set. However, in a source file you can have statements that include

other files (such as interface files). These include statements use either the <> syntax or the “” syntax around the name of the file you wish to include. The <> syntax tells CodeWarrior to look in the System set only. The User set is skipped completely.

The “” syntax refers to the User set. If you use that syntax in your source file, CodeWarrior looks in the User set first, as expected. If it does not find your specified file in the User set, it then looks in the System set folders.

If you turn on the Treat #include <...> as #include “...”option, seen at the top of the panel in Figure 3.21, CodeWarrior looks in both the User set and the System set for include files specified with <>. This may make the file search take longer, but is more thorough.

The bottom line to all this is that you can control the areas of your hard drive where CodeWarrior searches for files. This can make CodeWarrior more responsive and save you some time by limiting search areas. You can also make sure that if two files with the same name exist in different directories (not a wise idea, but certainly possible), you can control which one CodeWarrior finds first. If two files with the same name exist in the search path, CodeWarrior uses the one it finds first. You will not be warned that another file exists with the same name.

You should aslo know about a few other features of the Access Paths panel, but I’ll wait and revisit this panel when I discuss compiling and linking in Chapter 7.

## Customizing Stationery

Now that you’ve almost finished this section on the project manager, you’re aware that a project has dozens of different options you can set. In fact, a large part of this book is about what those options are, what they do, and how you can use them. In addition, many projects that

are aimed at a common target—for example, all your Windows application projects—share many common files.

Early in this chapter you saw that when you create a new project you have the opportunity to use an existing template project as the basis for the new project. You've already seen what stationery is, and how simple it is to use—you just pick the template from the New Project dialog.

CodeWarrior comes with a wide selection of stationery projects that get you started programming for your desired target. For example, for Windows 95/NT programming, stationery projects are available for a Windows application, a DLL, and a simple, console-based application. Similar stationery projects exist for Mac OS programming and Java programming. Other Metrowerks products designed for other targets (such as programming for the PlayStation game console) also come with stationery appropriate for the target.

These projects already have the correct target settings and preferences, the correct libraries included, and some generic source files so you can compile and build the project without making a single change. Of course, what you're really getting is Metrowerks' standard, lowest-common-denominator generic project. It is quite likely that you'll want to use different stationery than the standard templates that come with CodeWarrior. You may want to modify some settings, add libraries, remove libraries, or perform other tasks to truly customize the project for your individual needs.

So, how do you create your own custom stationery? It's really easy.

First you set up a project exactly as you want it to be. You can start with an empty project, or use an existing template as your starting point.

Remember, this is going to be the basis for future projects, so you can make it as specific or generic as you want. You can add files and libraries, and set all the options so that they reflect the way you like to run things. You can create placeholder or dummy files that you'll ulti-

mately replace when you build new projects based on the template. You can control what your target is, what languages you are using, how the compiler and linker work, what colors to use for keywords in the source code, and much more.

**NOTE**

You probably don't know how to do some of these things yet, but after you have accomplished these tasks I'll remind you that you can save all your work as a stationery project.

For example, you might create a project named My C App that you would use as a template for standard Windows application using the C language, with your own preferred compiler settings. You might include default or common source files, documentation files, and so forth. For example, you might have a short text file that has a standard copyright notice you like to copy into every source file. You could add such a text file to your stationery project so it will always be there when you start a new project.

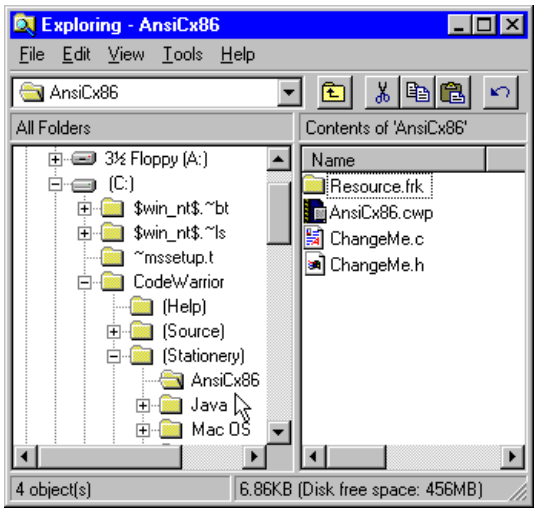
Once the default, generic project is set up exactly as you like it, you can turn the project into stationery.

When you finish setting up your complete project, simply close the project. Go to the Windows desktop, and make sure that your project folder contains all the files you want as part of your template. Then move the complete folder into the **(Stationery)** folder. The next time you create a new project using this template, your new project folder will contain not only the project file, but copies of all the other files as well. Figure 3.22 shows what such a folder might look like if you explored inside the **(Stationery)** hierarchy.

You can have as many different stationery projects as you like. Ultimately, only you know the kinds of projects you have, and how you want to build them. For any kind of project that you expect to reuse, create stationery for it to save yourself some work.

FIGURE 3.22

You can put many files in a stationery folder and all will be duplicated in a new project.




Within the **(Stationery)** folder, you can organize the stationery files any way you want. Each project file must be in its own folder, but beyond that you are free to store them any way you wish.

**NOTE**

The parentheses in the name of the **(Stationery)** folder are necessary to prevent the compiler from looking inside this folder for source files. This is important, because the folder contains actual source files that CodeWarrior uses when creating a new project. After you create a project, you are sure to change these files. You wouldn't want the compiler going after the wrong files and using the original rather than your new copy.

You have absorbed a tremendous amount of theoretical information about how the CodeWarrior project manager works. Now it's time for some hands-on experience.

## Hands-On Exercises

Because this is the first exercise in which you actually start working with a project, let's talk for just a minute about how these exercises work, and how you should follow instructions.

Each exercise in this book has an accompanying folder containing a sample project and code. Each folder is named Exer*nn*, where "*nn*" is the number of the chapter. In most cases, you start the exercise by opening the CodeWarrior project that's in the exercise folder. However, in the exercise for this chapter you'll create a new project from scratch.

Because this book is about CodeWarrior tools, not about how to write code, some chapters contain a single project and set of source files you use for exploration and experimentation. However, sometimes you will write actual code, and for those exercises you'll find two projects—one each in folders named **Start Code** and **Solution Code**. When this happens, start with the starter project, and use the solution project if you run into trouble.

Because very few of the exercises (or the steps in an exercise) have you writing or modifying code, you can take certain liberties. If you feel like using a different file arrangement or names for your projects, go ahead and do so. You start each chapter with a fresh project and fresh code, so you won't go too far astray. If you encounter problems as a result of your adventurous spirit, you can always come back to the precise instructions and follow them carefully.

Make sure, however, that you read and understand each step in each exercise, because they have been designed to take you through every major part of CodeWarrior logically and methodically. Still, if you're a free spirit who likes to run with an idea, and you see a good one come your way, go for it. Just keep in mind, if the step actually has you writing code, you had better write the code *exactly* as described in the step. Other parts of the entire project (most of which is given to you in the start code) may depend upon the precise names of variables, functions, and so forth.

So, if it involves code, follow the instructions to the letter—you know how precise code syntax must be. Otherwise, you can enjoy some latitude and use the steps as a jumping-off point for your own explorations.

With Step 1, you'll learn how to create a new project. In this exercise you're going to create stationery for an ANSI C project that does not use the Windows API. Then you'll use that as the basis for a new project.

**WARNING**

The exercises in this book assume that you are using a full version of CodeWarrior. CodeWarrior Lite (which is included with this book) does *not* allow you to create a new project, create a new source file, add or remove files from an existing project, or save changes to an existing file. Many steps in the exercises in this book require that you perform one or more of these tasks. Therefore, if you are using CodeWarrior Lite, use the solution code for the exercise. The solution code project already has all the files in place, and all the code ready to go. You can read and follow along in the exercise, and still complete the work (such as compiling and running the code) using CodeWarrior Lite.

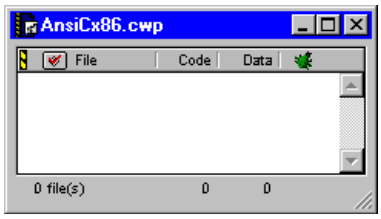
**1. Create a new empty project.** If CodeWarrior is not already running, launch it. Choose the New Project item from the File menu (Shift+Ctrl+N) to create a new project. In the New Project dialog box, choose to create an empty project, leave the Create Folder checkbox on, and then click OK to accept the stationery choice.

The Name New Project As dialog opens so you can save your new project. Navigate to the **Exer03** folder on your hard drive and open it so you can save your project inside this folder. Then give your project a name. The solution code for this chapter is named **AnsiCx86.cwp**. When you're all done, click the Save button to create the new project, and a new Project window appears, as shown in Figure 3.23.

You'll add files and work with them in this exercise. To get ready for this (not a task you would typically have to perform when working

FIGURE 3.23

A new and empty project awaits you.




with CodeWarrior), go to the Windows Desktop to see what things look like. Locate the **Exer03** folder and open it.

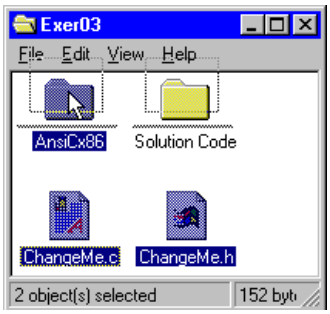
Inside you'll see the **Solution Code** folder for this exercise, and a new folder that you just created named **AnsiCx86**, or whatever you named your project. You'll also see two files named **ChangeMe.c** and **ChangeMe.h**. Move these files into the new project folder. Figure 3.24 shows what this looks like. You'll be using these files with the stationery project you're building, and they should be in the same folder as the project to make subsequent steps easier.

**2. Explore the project settings.** If you are still on the Windows desktop, return to CodeWarrior to continue with the exercise. Once you are back, looking at the empty Project window, the first thing you should do with a new project is set its target. Choose the Project Settings item in the Edit menu to display the Settings dialog, and then choose the Target panel, which is shown in Figure 3.18.

If it is not already selected, choose the Win32x86 Linker target. You have just told CodeWarrior that the code you are going to write is intended to run under Windows 95 or Windows NT.

The file types listed in the Target panel are default file types recognized for this project. They'll do just fine. However, take a look at the list to see the different extensions and the compiler associated with each.

FIGURE 3.24
Move two files into the project folder for later use.




While you're here, you should examine a few more panels and change some other settings. Take a look in turn at:

- x86 Project
- x86 Linker
- C/C++ Compiler
- Build Extras
- Access Paths

In the x86 Project settings, you can set the name of the application to something better than **NoName.exe**. You won't actually build an application until the next chapter, but this is where you set your final product's name. You should also note the output file type, where you decide whether to build an executable, a library, or a DLL. Leave this as an application for this project.

## The Thought behind the Resource Fork Shadow

In a folder where you have created a CodeWarrior project, you'll see a subfolder named **Resource.frk**. If you look inside this folder, you'll find one or more files with the exact same name as files in your project folder. You might wonder what's going on here.

What you're seeing is an echo from the Mac OS file system, in which a single file consists of two parts, or forks. One is called the *data fork*, the other is called the *resource fork* (hence the folder name, **Resource. frk**). Because of its Mac OS heritage, CodeWarrior expects to find certain information inside a file's resource fork, which doesn't exist in Windows. To mimic this behavior (so it can run successfully and exchange files between platforms), CodeWarrior creates a *shadow* of each file that would have a resource fork, and places it in the **Resource.frk** folder. This file holds data in specific formats. For example, the resource fork shadow of a source file might store the font you want to use for that file.

The resource fork shadow of any file is created and maintained automatically without any intervention, action, or effect on you, with one exception. What was effectively one Mac OS file is two files on Windows. If you decide to move a file from one folder to another, you must bring its resource fork shadow along with it, and place it in a **Resource.fr**k folder at the new destination.

If you do not bring the shadow along with you, unpleasant things may happen. For example, if you fail to bring a project file's shadow from the resource fork folder, you won't be able to open the project. So the best advice is, once you create a file, leave it where you put it. If you must move it, remember to look in the **Resource.frk** folder for the other part of the file.

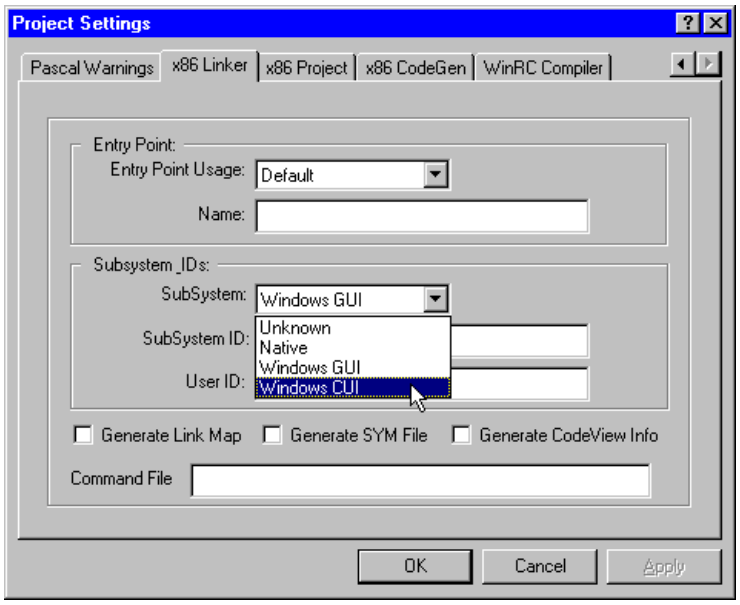
There's no need to move source file shadows in Step 1, because the files will work without the **Resource.frk** shadow. This is most definitely *not* true of project files. If you move a project file from one place to another manually, you must move its **Resource.frk** shadow along with it.

Now go to the x86 Linker panel, shown in Figure 3.25. In this panel, note the SubSystem pop-up menu in the center of the panel. From that menu, choose the Windows CUI as the target subsystem. This is the Console User Interface, as distinguished from the Windows GUI.

Next, go to the C/C++ Compiler panel. At the bottom of this panel is a field for the prefix file. I'll talk about prefix files in detail later on.

FIGURE 3.25

The Console User Interface subsystem is used for creating applications that use console I/O.




For now, simply delete any file name that appears in this field so that there will be no prefix file.

Also examine the other two panels listed above. The default values in these panels will do fine for this exercise.

When you are through, click the OK button to save your settings in all the panels. You have picked a target, and have specified the kind of project you're building (an application), and that it uses console I/O rather than the Windows GUI.

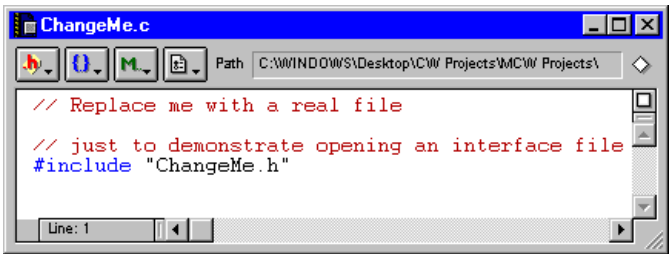
**3. Add a file to the project.** In this step you add the file contained in an open window directly to the project. This is the simplest way to get a file you're working on into a project.

First, use the Open command in the File menu to open the **ChangeMe.c** file. You'll find this file in the same folder as your project, where you

moved it back in Step 1. When you have the window open, you'll see (as shown in Figure 3.26) that it has two comments, and includes a header file. This is not a real source file but is simply a placeholder file. What you're doing is creating a project you can use as stationery later on.

FIGURE 3.26

The **ChangeMe.c** file is a simple placeholder you can use in stationery.



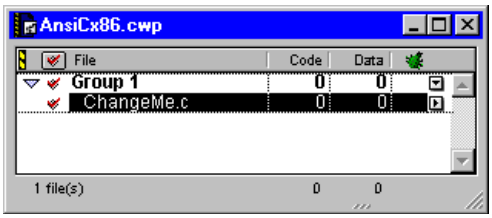

Now that the window is open, use the Add Window item in the Project menu to add this file to the project. (If the Add Window item isn't available, close the window, and then use the Add Files item discussed in Step 4 below.) When you're done, the Project window looks like Figure 3.27. Congratulations! You've started building your first project.

Close the **ChangeMe.c** file when you've completed this step.

**4. Add multiple files to the project.** In this step you add a series of library files to the project. These are the libraries you would use if you were creating a simple ANSI C console application, not one that uses the Windows API. This conforms to the choice of the Windows CUI subsystem back in Step 2.

To begin this step, use the Add Files item from the Project menu. In the resulting dialog box, navigate to the **\CodeWarrior\Metrowerks Standard Library\MSL C** folder and use the dialog box to select the following library file:

FIGURE 3.27
The Project window after you have added your first file.




- **Ansicx86.lib**—the Metrowerks standard C library

Then repeat the process and navigate to **CodeWarrior\Win32-x86 Support\Libraries** and add this file:

- **Mwcrtl.lib**—Metrowerks C runtime library

This technique works just fine for source files as well as libraries, but in this case you're adding library files. When both files have been selected, click the Add button to add the files to the project. When you do, you may see a window appear with a message that an access path has been added to the project. That's because these files are in a folder new to this project. CodeWarrior takes care of the access path automatically.

Repeat this process one more time to locate and add three more library files in the to **\CodeWarrior\Win32-x86 Support\Libraries\Win32 SDK** folder, listed here.

- **Gdi32.lib**—the Windows graphics library
- **Kernel32.lib**—the Windows kernel library
- **User32.lib**—the Windows user interface library

Although you aren't building a Windows application, the console interface lives in the Windows environment, and the code in these libraries is necessary even when creating a simple ANSI application.

When you have added all the files, the Project window should look like Figure 3.28.

FIGURE 3.28

A project with all the necessary libraries.

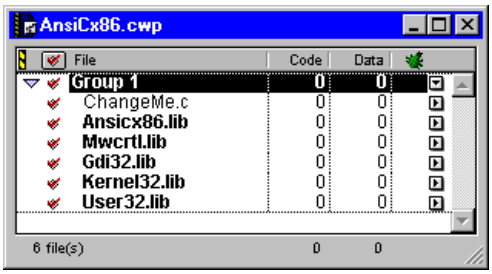
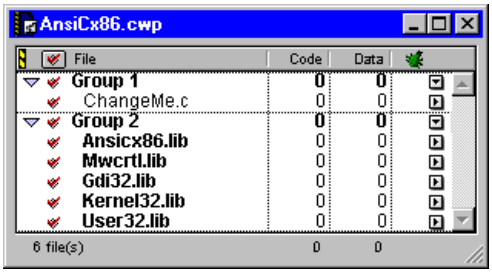



FIGURE 3.29

A Project window after you create a second group



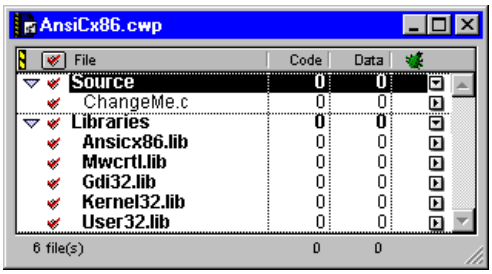

**5. Create a group of files.** In Figure 3.28 you can see a single group of files, named Group 1 by default. In this step you create a new group to keep the libraries together. The purpose of this step is purely organizational, and has no impact on the final project. You can keep all your files in a single group if you wish.

To complete this step, select the first library file after the source file. You can do this using the arrow keys, or simply point and click. When the file is selected, it's highlighted. Then press Ctrl+Enter to create a new group, named Group 2, with that library as the first file. Figure 3.29 shows the result.

**6. Name the groups.** To modify the name of a group, double-click the group name, or select it and press Enter. In the resulting dialog box, type in a new name. You might rename Group 1 as "Source" and Group 2 as "Libraries", as shown in Figure 3.30.

FIGURE 3.30

Give groups logical names for your file organization scheme.

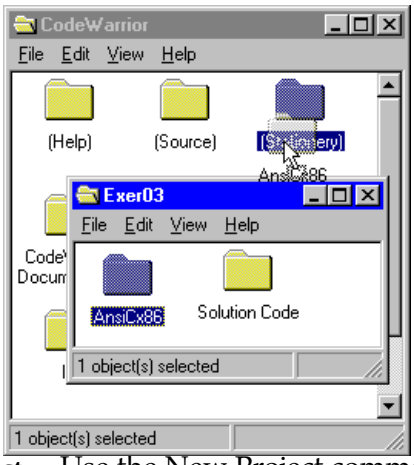



**7. Save the project as stationery.** You now have a complete project, with a target selected, project settings modified, and groups of files. Close the project, and go to the Windows desktop. Locate the folder that contains this new project, and physically move it from the Exer03 folder into the **\CodeWarrior\(Stationery)** folder. You can put it anywhere inside the **(Stationery)** folder you'd like, but for our purposes you can just drop it right into the **(Stationery)** folder as shown in Figure 3.31.

Now that you have created some stationery, it's time to use it. In the next step you create a new project based on this stationery, and then work with the files in the project.

FIGURE 3.31

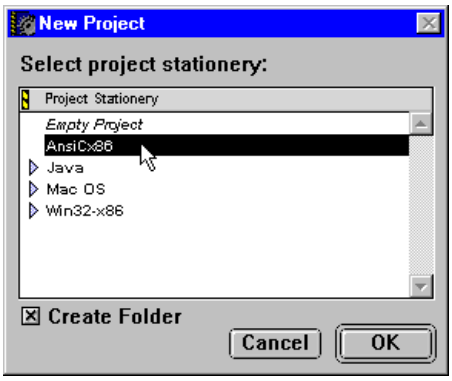
Put the new project in the **(Stationery)** folder to turn it into a project template.




**8. Create a new project.** Use the New Project command to display the New Project dialog, as in Step 1. This time, the list of available stationery includes the **AnsiCx86** project, as shown in Figure 3.32.

FIGURE 3.32

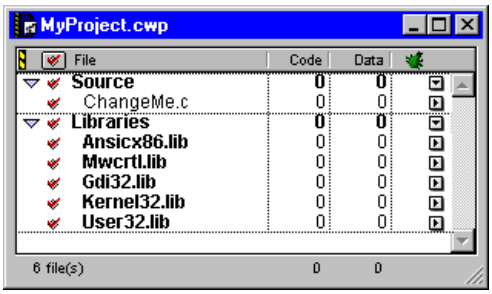
You can use your own stationery as a template for a new project.




Select **AnsiCx86** as the template for your new project, and save the new project back in the **Exer03** folder. You might name it **MyProject.cwp**. When you save the new project, it appears in CodeWarrior, and looks just

FIGURE 3.33

A project based on stationery is identical to the original template.




like the original stationery. Compare Figure 3.33 with Figure 3.30; but for the name of the project they are identical.

Even better, go to the Windows desktop and look inside the folder for this new project folder inside the **Exer03** folder. And there you'll see copies of the original source files, all ready for your use. Now return to CodeWarrior for the rest of this exercise.

**9. Bring the project up to date.** Although the code in **ChangeMe.c** doesn't do anything except include an empty header file, you can still compile the project, and you have to so that the project knows which files are included in your source code. In a subsequent step you'll be opening included files, and that won't work unless the project has been compiled.

To accomplish this step, choose the Bring Up To Date item in the Project menu. A bunch of stuff happens at this point. First, all the files in the project are compiled. For the libraries, that means that their code is brought into the project. The numbers for Code and Data in the Project window will change, as shown in Figure 3.34. The numbers for the **ChangeMe.c** file are still zero, because no real code exists in that file.

**10. Open a source file.** There's only one file that can be opened in the project, and that's the **ChangeMe.c** file. Double-click the name in the

**WARNING**

If you choose Make, the project will not link and you'll get an error because there is no **main()** function anywhere in this code. It's fine to ignore that for now. If this happens, just close the error message window.

Project window to open the file, or select it and press Enter. The source file opens in an Editor window. You'll meet the editor in the next chapter.

**11. Open an interface file.** Although included files are not listed in the project, you can open them very easily. Here are two ways you can do it.

In the Project window, click on the pop-up menu to the right of the **ChangeMe.c** file, and you'll see the menu as shown in Figure 3.35. Two items appear in this menu for this file. You can touch the **ChangeMe.c** file, or open **ChangeMe.h**. Open the **ChangeMe.h** file, and then close it again so you can practice another technique.

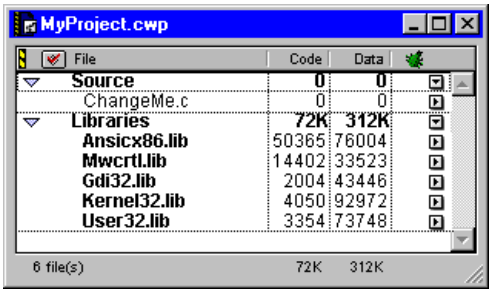
If it isn't already open, open the **ChangeMe.c** source file from the project. In that file, you'll see the following line of code:

**#include "ChangeMe.h"**

Select the word "ChangeMe" by double-clicking on it, and then press Ctrl+D (the Open Selection item in the File menu). The selected

FIGURE 3.34

Bringing a project up to date compiles all the code.



file opens. Note that you didn't have to select the file extension, just the name. CodeWarrior takes care of the extension automatically.

**TIP** If you type Ctrl+Tab you cycle through all open windows. Try it!

**12. Remove binaries from a project.** To complete this step, choose the Remove Binaries item from the Project menu. (Be careful you don't choose Remove Files!) You'll get a warning message, as shown in Figure 3.36. Click OK to remove the object code and streamline the project. Notice that all the values in the Code and Data columns in the Project window revert to zero.

**13. Close the project.** Now, close all the windows. You can Alt+click in the close box to close all open editor windows, leaving only the Project window open. Click its close box as well. CodeWarrior saves the project automatically.

FIGURE 3.35

Use the file pop-up menu in the Project window to open an included file.

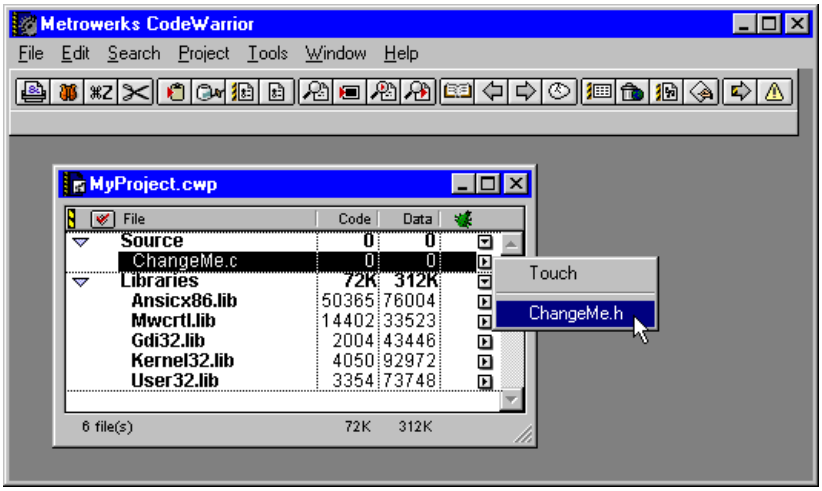
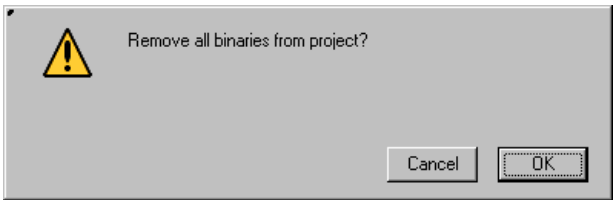


FIGURE 3.36

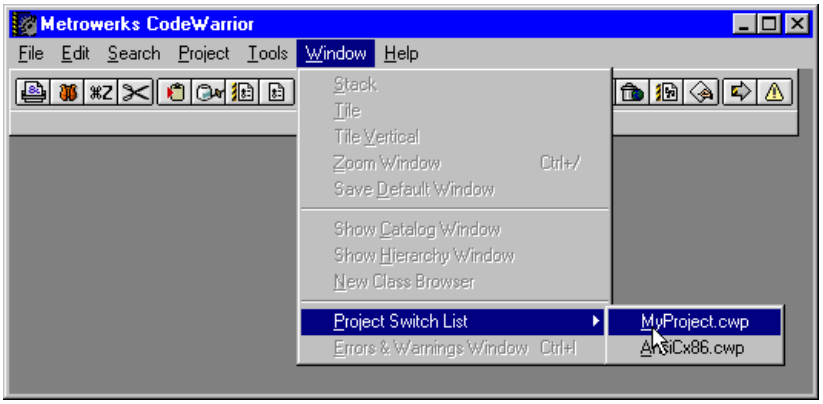
You must confirm removing binaries because it means you must recompile the entire project.




**14. Open the project with switch list.** A very slick way to get to a previous project is to use the Project Switch List. Look in the Window menu for that item, shown in Figure 3.37. As you can see, both projects you have been working on—the stationery project and the new project—appear in this list.

FIGURE 3.37

The Project Switch List contains all recently open projects, so you can go right to the project you want.




Select your project from the list and it opens. When you are through, you can close the project file again. You've now finished the exercise. If you'd like to continue exploring, feel free to do so. In the next chapter, you'll write some code and run a real application.

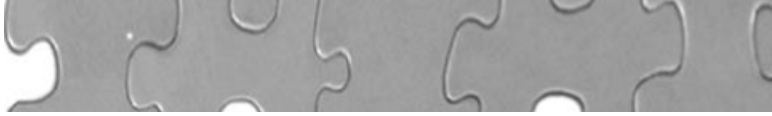
## Summary

Although this is only the first substantive chapter in which you gain real experience with CodeWarrior, you have accomplished a great deal. You have already come a long way toward mastery of the CodeWarrior tools.

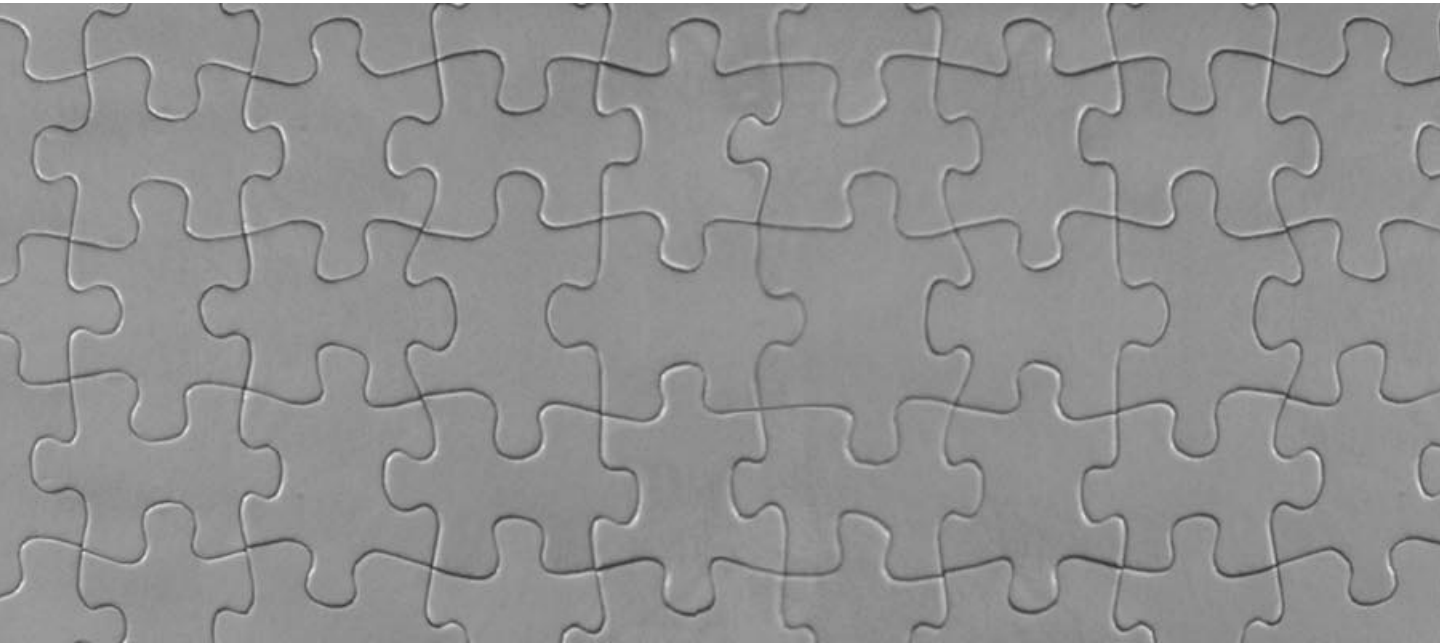
You understand the project concept and the CodeWarrior project manager. You know how to create a new project and how to use and create project stationery. You can add and remove files and groups of files from a project, and you've learned how to open files and navigate among them using the project manager.

You also know how to control compiling and debugging from the project manager. Finally, you saw how CodeWarrior manages preferences and settings. You can manage both the back-end target compilers and the front-end language compilers.

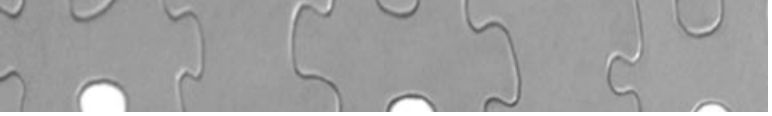
Best of all, this information is fundamental to CodeWarrior and will serve you well throughout the rest of the book. If you feel a little unsure of your mastery of the project manager, don't worry—you will have many opportunities in the ensuing chapters to practice your skills. The project manager is the core of CodeWarrior, and the knowledge you have already gained forms the core of your expertise in CodeWarrior. Now let's look at how to work with individual files in CodeWarrior.



# PART



# III



# Using the Core Tools

This part of the book teaches you how to work with CodeWarrior's core tools.

In Chapter 4 you meet the source code editor. This is where you write code. What's more, the editor provides you with some excellent and unique features for navigating through your code.

Chapter 5 covers the editor's powerful search and replace capabilities. You'll learn how to search and replace code in a single and multiple files. Chapter 6 explains the integrated browser. The browser is a world-class interface for navigating quickly and easily through your entire code base.

Of course, at some point you must compile and link your code—Chapter 7 shows you how. Chapter 8 covers the debugger in detail—and the CodeWarrior debugger packs a lot of power.

After completing this part of the book, you will be well on your way to mastering CodeWarrior.

# CHAPTER FOUR

4

# Editing Your Code

- Creating a New File
- Opening an Existing File
- Editing Source Code
- Cool Features
- Saving a File
- Closing a File
- Printing a File
- Hands-On Exercises

Writing code is what programming is all about, and this chapter shows you how to write code in CodeWarrior. This chapter shows you how to use the editor to write and edit code, and how to navigate through your work quickly and easily. You will also learn how to create, open, save, close, and print individual source code files.

## The Fundamentals of Files and Windows

Source code must exist over an extended period of time. It cannot disappear when you shut down your computer or exit from CodeWarrior. To persist for the long term, you have to save your work in files. When you subsequently open a file, its contents appear in a window.

CodeWarrior, like most software, associates each window with a distinct file. For example, the project window represents the project file on disk. Source code, notes, resources, all can be stored in files, and each can be opened in a separate window in the CodeWarrior workspace.

The title bar of the window matches the name of the file on disk. When you open a window, its contents represent the data stored in the file on disk. You make changes to the contents of the window, and all is well.

Note, however, that you're changing the contents of the *window*, not the file. Because a close one-to-one relationship exists between a window and its file, it is easy to think of them as the same thing. In fact, most of the time that's a very safe assumption to make. But they are not the same, and forgetting that there is a difference between a window and a file can get you into trouble.

continued ➠

### The Fundamentals of Files and Windows

The changes you make in a window are not instantaneously stored in the file on disk. That doesn't happen unless you save the data. While you are working in the window, your changes exist only inside the perilous world of your computer's memory. If you lose power, for example, everything is lost.

While CodeWarrior can't ensure a steady power supply, it can protect you from unintentionally closing a file and losing your work. In any circumstance, if you attempt to close a window that has unsaved changes, you will see a warning that alerts you to the danger and asks you whether you want to save or discard the work.

So as you work with windows and files, don't forget the distinction between the two and save your work often.

## Creating a New File

The first step toward creating a new file is to create a new, empty window. After that you can write code in the window, and save the code to a new file. To create a new window, choose the New item from the File menu. When you do, CodeWarrior opens an untitled window in the source code editor. Figure 4.1 shows such a window.

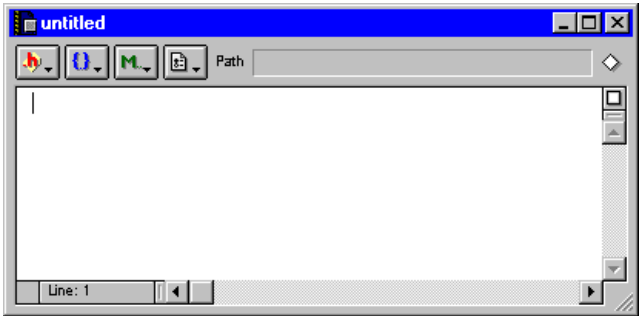
One quick look at Figure 4.1 tells you that a lot is going on with an editor window. I'll discuss all the features in this window in just a bit. The point here is that this window is the first step toward creating your own source code.

**NOTE**

Because this is a new window, no file exists yet. That's why the window is untitled. When you save the window, CodeWarrior creates the associated file, writes the contents of the window to that file, and renames the window to match the file name.

FIGURE 4.1

A new and empty Editor window ready for the product of your genius.




## Opening an Existing File

While you'll create new files on a regular basis, often you'll want to open an existing file to see, use, or modify the contents. CodeWarrior has several mechanisms for opening existing files. In the previous chapter you learned some ways to do this. For example, you can use the Open command in the File menu to open any text file. However, you have several easier ways to open project files. You can also:

- Double-click a file name in the project window
- Select one or more files in the project window and press the Enter key
- Use the File pop-up menu to the right of each group name in the project window
- Use the file pop-up menu to the right of each source file's name in the project window to open any included interface file

**TIP**

The File menu also has an Open File command. If you choose this item, a dialog box appears in which you can enter the name of a file. CodeWarrior then searches through the project's access paths, finds the file, and opens it. This procedure saves you from hunting for a file by navigating the standard file dialog box.

You have still more ways you can open files connected to a project. The editor, the browser, and the debugger all have powerful and intuitive mechanisms for opening the files you want to see. You'll meet them as we discuss each tool in turn. In some cases, you don't even have to specify the file you want. CodeWarrior knows which file contains the code you're interested in, and opens it for you. Many discoveries await you on the journey ahead.

However, now that you have a file open, you can take the next step on the journey: filling the file with code.

## Editing Source Code

You use the source code editor to write new code and modify existing code. At its simplest, the source code editor is a specialized text processor. In fact, you can open any text file with CodeWarrior, not just files that follow the syntax rules of a particular programming language.

Calling the editor a simple text processor hardly does it justice, however. The source code editor has an extraordinarily rich feature set designed for the kinds of tasks you perform as a programmer. This section covers some of the editor's basic features. A liitle later on, in the Cool Features section, you'll encounter advanced features that make the CodeWarrior editor especially useful.

### The Thought behind the Editor

The source code editor grew out of years of programming experience. The CodeWarrior engineers are a seasoned group of veterans with decades of real-world experience. They have been using source code editors for all that time, and know from the user's perspective what features are vital.

continued ➠

### The Thought behind the Editor

In addition, the Metrowerks motto is, "We listen and we act." Metrowerks pays close attention to its customers, and regularly adds features to the editor that have been widely requested by real programmers. As a result, you are quite likely to find some unique or interesting feature that brings a smile to your face. This attitude toward customers—that the company and the customer are part of the same team—means that the tools are more likely to have the feature set you need.

The engineers have tried to put the best features together in one place to make your programming tasks as simple as possible. You'll see a heavy emphasis on navigation—the ability to move around a file and between files quickly and easily. This is, after all, where many programmers lose a lot of time. The tools can't write your code for you, but they can get you to where you want to go nonstop.

The editor is not finished, and probably never will be. At the moment, perhaps the most widely requested feature not yet implemented in the editor is a file comparison feature. It's in the works and should appear before too long.

The entire CodeWarrior tool set, not just the editor, is a work in progress. As the needs of programmers evolve, so too does CodeWarrior.

## Understanding the Editor Window

Before we discuss how to work with the editor, let's take a quick look at the components and controls that make up the Editor window. Figure 4.2 illustrates the window and its various controls.

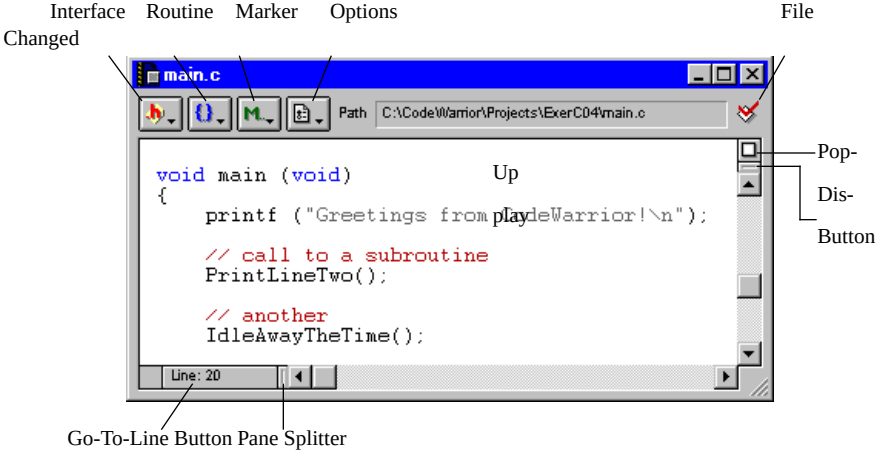
You'll meet and use each of these controls in this chapter. In the rest of this section, I'll talk about how to perform standard editing tasks such as:

- Adding text
- Moving the insertion point

- Selecting text
- Moving text
- Deleting text

These are all very common features of any text editor, and you might be tempted to skip the discussion. However, you might find some interesting tidbits here. For example, the editor supports drag-and-drop for moving text, dynamic scrolling, and has useful keyboard navigation features for moving the insertion point.

FIGURE 4.2
An Editor window has many tools you can use to make your work easier.




So, while some of this information is admittedly basic (and I'll try to get you through that as quickly as possible), some finer points exist also. A little later on, in the "Cool Features" section, you'll explore how to use the buttons and controls in this window to get your work done as efficiently as possible.

## Adding Text

While the editor has many features related to programming, at its core the editor is in fact a text processor. To add text, just type. As you type, your code appears at the insertion point—the flashing cursor.

The editor does not wrap text. You can extend a line to an arbitrary length to the right. CodeWarrior begins a new line when you press the Enter key. This is the approach that makes the most sense for source code. You want absolute control over what code appears on what line at all times. CodeWarrior gives you that control.

## Moving the Insertion Point

One simple task you often do when working with code is to move the insertion point. In fact, you do this task so often you may not even realize how automatic it becomes. CodeWarrior provides several standard mechanisms you can use to accomplish your goal. In addition to the standard mouse and arrow-key mechanisms for moving the insertion point, CodeWarrior has a couple of neat tricks. You can pick which techniques work best for you.

### The Fundamentals of Setting the Insertion Point

Unless you're a first-time computer user, you already know this stuff. You can set the insertion point using the mouse. Simply click where you want the insertion point to be. If the spot isn't visible, use the scroll bar controls to move the text until the desired location appears in the window.

You can also set the insertion point using the keyboard. You control the insertion point and scroll text using the cursor control (arrow) keys. Use the Left or Right arrow key to move one character to the left or right. Use the Up or Down arrow key to move the insertion point one line up or down. Text scrolls automatically so the insertion point remains visible at all times.

If you press the Ctrl key along with an arrow key, you move the insertion point to the beginning or end of the line, or to the top or bottom of the entire file. Table 4.1 summarizes how the insertion point moves when you press an arrow key alone, or with the Ctrl key. Using

the arrow keys, you can quickly scroll around your text without ever leaving the keyboard to use the mouse.

TABLE 4.1 Insertion point movement with arrow keys

| Key | No Modifier | Ctrl |
|---|---|---|
| left | previous character | beginning of line |
| right | next character | end of line |
| up | line above | top of file |
| down | line below | bottom of file |

**TIP**

You can use the Page Up, Page Down, Home, and End keys to scroll around the window. These keys do not move the insertion point, however. If you are scrolled away from the insertion point, you can return instantly by typing an arrow key.

To move the insertion point to a specific line, click the line number at the bottom of the editor window. A dialog box appears asking you for the desired line number, as shown in Figure 4.3.

**NOTE**

Other more powerful and useful ways to move around also exist, but I'll reserve discussion of them for the "Cool Features" section later in this chapter.

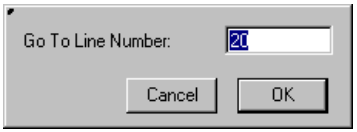



FIGURE 4.3
The Go To Line Number dialog box lets you jump directly to any line in a file.

## CodeWarrior as a Human Interface Model

As you have already discovered, and will see again and again throughout this book, CodeWarrior regularly provides an assortment of separate paths to a single goal. Moving the insertion point is a good example. You can use the keyboard or the mouse in a variety of ways to perform specific insertion-point-related tasks.

The toolbar concept is another good example. For most major commands in CodeWarrior, you can use a key command from the keyboard, a menu command chosen with the mouse, or click a toolbar icon. That gives you three completely different ways to do almost anything.

The net effect of this approach to human interface design is that CodeWarrior provides you with a web of possibilities. You choose which particular strand of the web to follow at any moment. You can mix and match different techniques freely, and without limit. Not only that, for the most part you can perform individual commands in any order as suits your purpose.

At first, the choices can be overwhelming. Over time, as you become accustomed to the tools, you will find yourself using a mix of techniques that—unconsciously—you have selected to best fit your personal, unique, and individual working style. In effect, you have customized the software to suit you, rather than forcing yourself to adapt to the software's idiosyncratic mechanism for getting a task accomplished.

A lot of thought has gone into the CodeWarrior human interface. This kind of flexibility, and the consistency you see across all the CodeWarrior products throughout this book, are not accidental. They are intentional and purposeful. The purpose is to make the tools as easy to use as possible. CodeWarrior doesn't tell you the right way to do things; it presents you with a set of choices, any one of which could be perfectly reasonable to a given individual at any moment. You choose which is right for you.

When you create software for yourself, your customers, clients, or friends, this is what you want your software to be like. Isn't it great that when you are developing software, you have the same kind of power?

## Selecting Text

While working with code, you must frequently select text. CodeWarrior supports all the usual methods for text selection. As with cursor control, you can select text using either the mouse or the keyboard, or a combination of the two.

You can select an arbitrary range of text, an individual word, or an entire line using the mouse alone. To select a range, click and drag the mouse over the desired text range. Select a word by double-clicking the word. Select an entire line of code by triple-clicking the line. This last trick is a very nice feature.

The keyboard has similar functionality. Select a range of text by pressing the Shift key and using an arrow key. CodeWarrior extends the selection from the original insertion point. Use the Left or Right arrow key to add or remove characters from the selection. Use the Up or Down arrow key to add or remove entire lines. Select from the insertion point to the beginning or end of a line with Shift+Ctrl+arrow key (Left or Right).

You can also select a range of text using the keyboard and mouse simultaneously. Press the Shift key and click at a location in the source code. This is called *Shift+clicking*. The editor selects all the text between the insertion point and the mouse click. Table 4.2 summarizes the various text-selection techniques.

TABLE 4.2 Techniques for selecting text

| Select | Mouse | Keyboard | Mouse and Keyboard |
|---|---|---|---|
| Range | click and drag | Shift+arrow keys | Shift+click |
| Word | double-click | n/a | n/a |
| Line | triple-click | Shift+Ctrl+arrow keys | n/a |

The Ctrl key used in combination with the Shift key extends the selection in exactly the same way it moves the insertion point. For

example, you can select all text to the end of the file by pressing Shift+Ctrl+down arrow. Refer to Table 4.1 provided earlier to review how the Ctrl key affects insertion point movement.

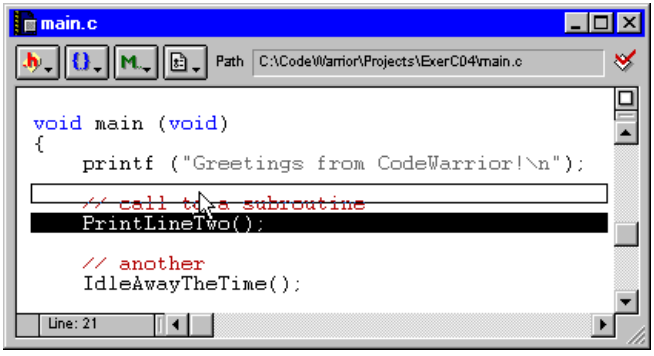
## Moving Text

Once you have selected text, you may want to move it around. CodeWarrior has two techniques for moving text. You choose either method: cut-and-paste or drag-and-drop.

Use the traditional cut and paste techniques by choosing the commands in the Edit menu. You can paste into the same window, or into any other editor window. You hardly need to be told about this, but I mention it here for completeness.

CodeWarrior also supports drag-and-drop for moving text around. After you have selected text, click anywhere in the selected area and drag the text to the location where you want to move it. As you move the text, the new insertion point flashes showing you where the text will appear when you drop it, as shown in Figure 4.4. When the insertion point is where you want it to be, release the mouse button. The text moves to the new location.

FIGURE 4.4
You can move text around using drag-and-drop.

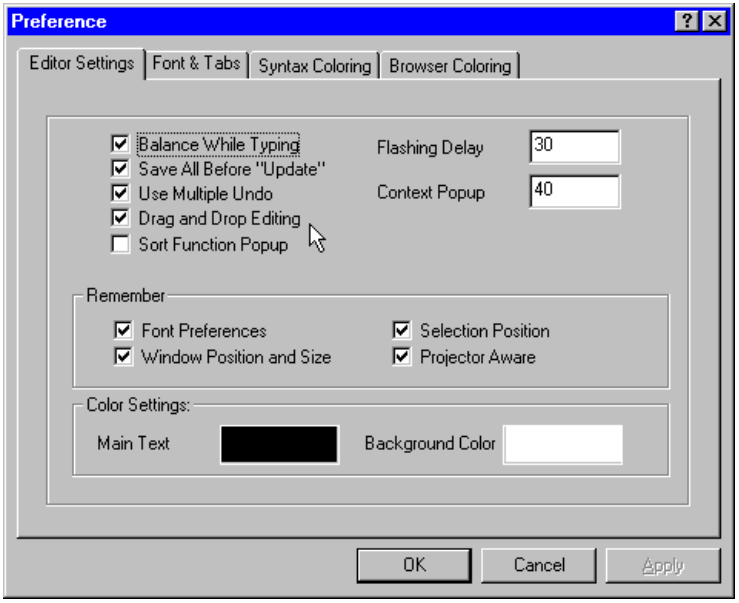


If drag-and-drop is operational, and you have a range of text selected, you can sometimes find yourself inadvertently dragging text. This happens when you click in the selected text, and then unintentionally move the mouse before you release the button. If this is a problem for you, and you don't like to use drag-and-drop for moving text, you can turn it off.

You control drag-and-drop in the Editor Settings preferences panel, as shown in Figure 4.5. This option is on by default. Click the Drag and Drop Editing check box to turn this feature on or off.

FIGURE 4.5

Use the Editor Settings preference panel to turn drag-and-drop editing on or off.




This particular panel is the control center for how the editor works, and you open it by choosing Preferences from the Edit menu. A quick look at Figure 4.5 is a good way to see some of the many features such as drag-and-drop that the editor offers you. You'll meet most of them in this chapter, and I'll regularly tell you that you can control how each feature works by coming to this panel.

## Deleting Text

Another basic editing task is deleting text. Once again, the CodeWarrior editor offers different mechanisms you can choose from. Of course the standard Backspace and Delete keys work as you would expect.

To delete a range of text, select it using any technique described previously, and then press either the Backspace or Delete key. You can also select text and use the Cut command from the Edit menu. This method removes the text from the window, but the text remains on CodeWarrior's clipboard until you perform another copy or cut operation.

## Undoing Changes

CodeWarrior remembers each action you perform and keeps these actions in an undo/redo queue. You can navigate back and forth through this action queue using the Undo and Redo commands in the Edit menu. (No toolbar icon exists for Redo.)

CodeWarrior's support for multiple undo/redo lets you undo a whole series of actions at any time. This is very useful if you decide you want to retrace your steps. Similarly, while you are backtracking, if you change your mind you can redo all your steps just as easily.

One nice feature of CodeWarrior is that the precise text for the Undo and Redo items varies depending upon the most recent action. When you perform an action, the text in the Undo item changes. For example, if the most recent action you performed was typing, the Undo item says "Undo Typing." If the most recent action was a paste operation, the item says "Undo Paste." The text of the menu item reflects what you did most recently, so you can know what you are about to undo.

When you undo an action, you move back up the queue one step. When you do so, the Undo menu item text changes so that it reflects the next item back in the action queue. So, no matter where you are in the action queue, you know the nature of the next undoable action. The Redo item text also changes to reflect the action you just undid.

Multiple undo is a powerful feature, but it has limits. After you undo one or more times, you have moved back up the action queue some distance. If you then perform a new action (other than undoing or redoing previous actions), the queue of actions from that point down is flushed. So if you undo a series of actions and then perform a new action, you cannot then redo the previously undone actions. They are lost forever. You can, however, undo the most recent action, and any other actions further back in the action queue.

Some programmers find the possibility of losing steps like this unnerving. As in most of its behaviors, CodeWarrior gives you a choice. By default, the CodeWarrior editor (and many other parts of CodeWarrior) supports multiple undo. If you turn off multiple undo, CodeWarrior supports single undo. With single undo, everything works more or less the same, but the action queue is one item long. You can undo or redo only the most recent action.

If you have multiple undo turned off, the Edit menu has a single item that toggles between Undo and Redo, depending upon whether the most recent action has been undone.

**WARNING**

The key equivalents for the Undo and Redo commands overlap with single and multiple undo. Ctrl+Z does both Undo and Redo for single undo, so the same key toggles between undone and redone. For multiple undo, Ctrl+Z does Undo, and Shift+Ctrl+Z does Redo. If you forget you are using multiple undo and press Ctrl+Z repeatedly (expecting to toggle back and forth), you can wipe out a whole series of actions very quickly. This would be a certifiable Bad Thing.

You control which undo mechanism you want to use (single or multiple) in the Editor Settings preferences panel, as shown in Figure 4.5.

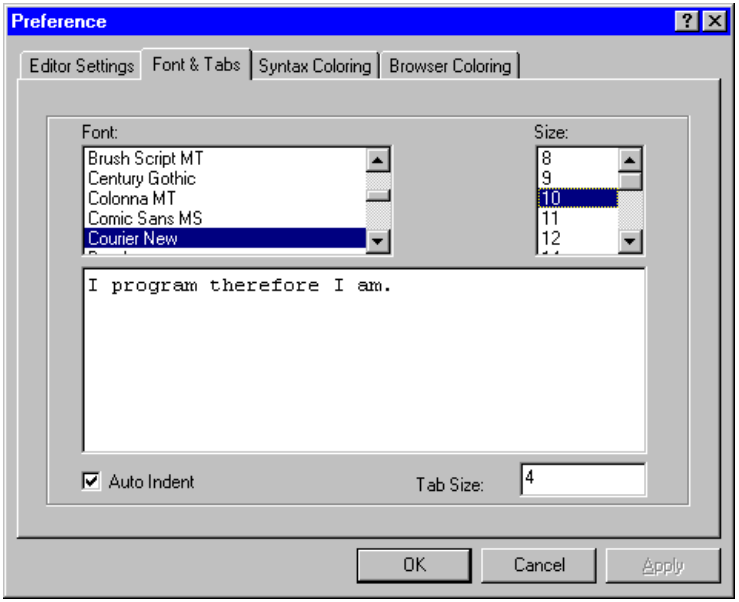
The factory (default) setting is for multiple undo to be on. Click the Use Multiple Undo check box to turn this feature on or off.

## Choosing Font and Size Preferences

You control the editor's choice of font and size in the editor's Font & Tabs preferences, as shown in Figure 4.6. Choose the font from the pop-up menu, and enter the size you want. The display area shows you a sample of the font and size you have chosen.

FIGURE 4.6

Set the editor's font and size in the Font & Tabs preference panel.




You use this same preferences panel to control how many spaces to use for a tab, and to control automatic indenting for code.

**TIP**

Always use a monospace font (such as Courier New) for code. Then everything always lines up nicely.

## Indenting Code

Most programmers like to use indented text to reflect different organization levels in code. For example, code that is subsidiary to a flow-control statement is indented to the right, as follows:

```
switch (inCommand)
{
    case kControlNext: // increment picture
        picture += 1;
        break;

    case kControlPrevious: // decrement picture
        picture -= 1;
        break;

    default:
        break;
{
```

When you press the Enter key, CodeWarrior moves the insertion point one line down and puts it at the left edge of the previous line. If that line has been indented, then the editor automatically supplies the necessary tabs or spaces so that your next line begins at the same place. This makes it easy to write a series of lines all indented at the same place.

You can turn off automatic indenting in the editor's Font & Tabs preferences, as shown in Figure 4.6. The Auto Indent check box is at the lower left of the panel.

You have one more way you can control indenting. You can select one or more lines of code and shift them left or right at will. Select a block of text, and then choose the Shift Left or the Shift Right command

from the Edit menu. The editor shifts all the text one tab stop to the left or right as desired. Use this trick to manually control the indent for an entire block of text.

# Cool Features

So far I have discussed the common or standard features of the editor you would expect to find in any code-oriented text processor. You have learned how to add, edit, move, delete, and navigate in your code. All these features are fairly straightforward. In this section you meet some of the editor's more advanced features.

Most of these cool features are related to various buttons and controls in the editor window. Each of these controls exists for a single purpose—to make you more productive.

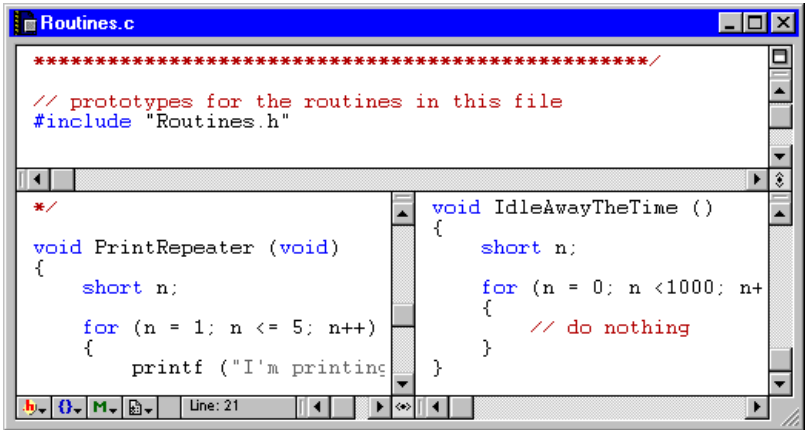
## Splitting Panes

How often have you wanted to look at two different parts of a file at once? A window just isn't big enough to see everything. CodeWarrior has a way around this problem. The editor lets you split any window into multiple panes, so you can see different parts of the same file simultaneously.

As with just about everything else in CodeWarrior, you have complete control. You can split a window horizontally or vertically, and split any resulting pane again and again in any arrangement that suits your fancy as shown in Figure 4.7.

You'll find one window pane splitter at the top of the vertical scroll bar. Just click and drag the control vertically and you split the window into top and bottom panes. You'll find the other splitter to the left of the horizontal scroll bar. Drag it horizontally and you split the window into left and right panes.

FIGURE 4.7
You can split any editor window into multiple panes.




Each pane has its own set of scroll bars and splitters, and can be controlled independently. You can scroll each pane to see different parts of the same file simultaneously. You can subdivide any pane into smaller panes, horizontally or vertically. As a result, you can see several different parts of the same file at once, each in its own pane.

No matter how many panes you have, only one insertion point is available, but that makes perfect sense. You can only edit text in one pane at a time. However, you can drag-and-drop text across panes, cut and paste, and perform all other editing functions regardless of the nature or number of panes you have.

You can switch panes by clicking the pane where you want to work. If you want to resize a pane, click and drag the pane resize box. To get rid of a pane completely, drag the divider to the edge of the pane to make it disappear.

**TIP**

You can do easy pane management with a quick double-click. To instantly split a pane into two equal parts, double-click a splitter control. You can get rid of a pane just as easily—double-click the resize box for that pane.

## Syntax Coloring

You decide what the basic text color and background color of editor windows are. You control these items in the editor's Editor Settings preferences panel, as shown previously in Figure 4.5.

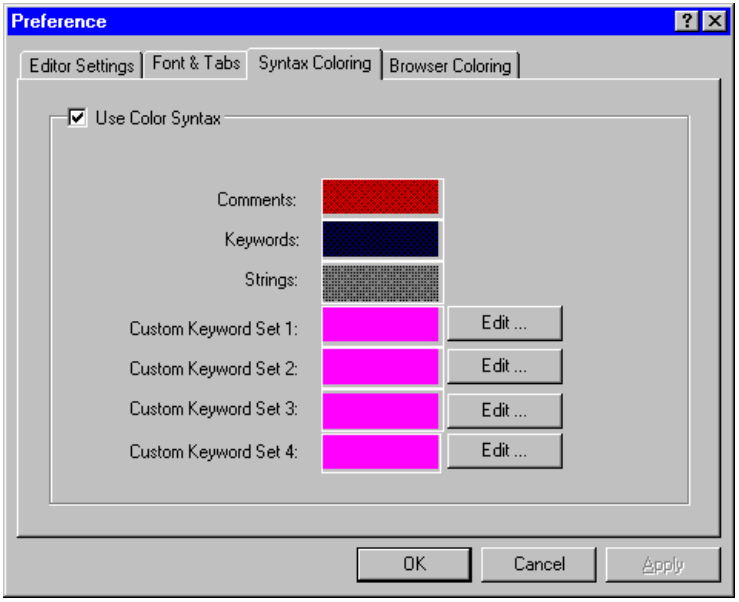
To change a color, simply click the corresponding color swatch. The standard color dialog box appears so you can set a new color. Because these are preferences, the colors you pick apply to all editor windows.

However, CodeWarrior gives you a lot more control than just setting basic text and background color. The CodeWarrior editor can use different colors to identify various elements of your code. Color is a great help in understanding your code. For example, you can make comments a special color, so they stand out and are immediately identifiable.

In addition to helping comprehension, using color can prevent mistakes. For example, if you program in C you are probably familiar with the problem of unterminated comments. You put in the starting syntax, /*, but you forget the closing syntax, */. When the compiler reads your code, it chokes and coughs up an error because whole chunks of code are interpreted as comments. If you use color syntax, the fact that a comment is unterminated becomes immediately apparent because all the code between the mistake and the next comment terminator changes color! You avoid this error because the color makes it easy to spot.

You activate syntax coloring and set the colors in the editor's Syntax Coloring preferences panel, shown in Figure 4.8. The editor automatically recognizes three items: comments, language keywords, and strings based on the standard syntax elements (such as comment delimiters and quote marks) used for these items. In addition, you can assign colors for up to four sets of custom words. When the editor encounters a word in one of these sets, it displays the word in the assigned color.

FIGURE 4.8
Use the Syntax Coloring preferences panel to control color for various syntax items.




You can set the color independently for each set of keywords. Just click the color panel for that item. Then the standard color dialog box appears so you can set a new color.

The list of custom keywords is completely modifiable. Click the Edit button to the right of the color panel. When you do, a dialog box appears in which you can edit the list, as shown in Figure 4.9. You can import a list from an external text file, or export the list to save it for future use. To add a word to the list, simply enter it in the text field and click the Add button. To remove an entry from the list, select it and press the Delete button. You can clear the entire list with the Clear button. Click the OK or Cancel buttons to accept or reject all changes.

Because these are preferences, the colors and custom lists you create apply to all projects. You can also assign colors on a per-project basis. If you want to have special color syntax for custom keywords in a particular project, choose the Project Settings command in the Edit menu,

and then select the editor's Custom Keywords settings panel. When you do, you see the panel illustrated in Figure 4.10.

FIGURE 4.9

Specify keywords that will appear in a unique color for all projects.

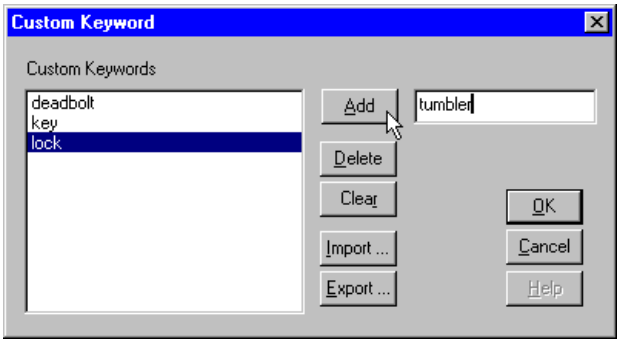
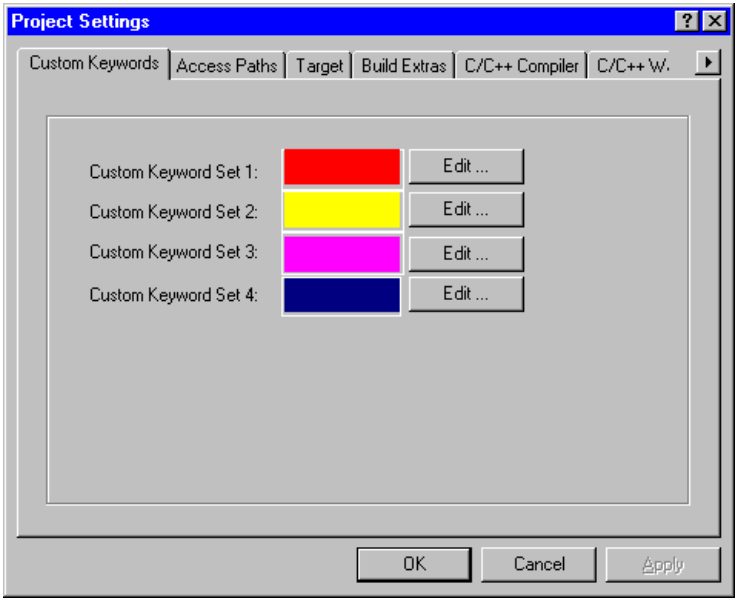



FIGURE 4.10

Use the Custom Keywords settings panel to specify custom keywords for a particular project.

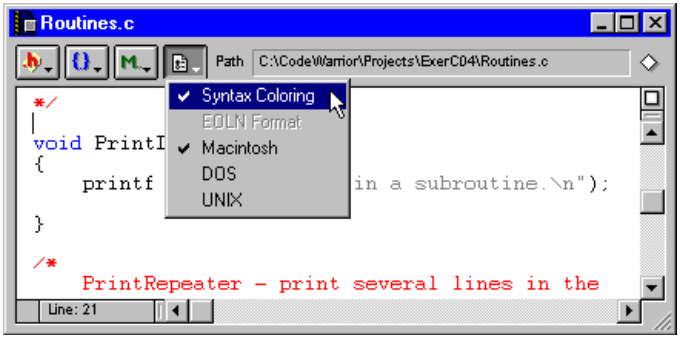


You can define four different sets of custom keywords for the project. You set the color for each set and modify each list exactly as you do for the global color syntax in the Preference window.

While you cannot turn syntax coloring off for a particular project (it is a global preference), you can turn it off for a particular file. Use the Options pop-up menu in the editor window, as shown in Figure 4.11. The first item is Syntax Coloring. Choose this item to turn syntax coloring on and off for the particular file. The syntax coloring state (on or off) is saved with the file and restored when you reopen the file.

FIGURE 4.11

You can turn color syntax off and on for a single file with the Options pop-up menu.




**TIP**

The browser has its own additional color settings for the different kinds of symbols it recognizes, such as enums, constants, class names, function names, and so forth. You'll learn how to control browser colors in Chapter 6.

## Balancing Punctuation

Like an unterminated comment, an unbalanced brace, bracket, or parenthesis—the {, [, and ( punctuation symbols—can be a real problem. If each opening punctuation mark is not paired with a matching closing punctuation mark, the compiler chokes and you get an error.

The CodeWarrior editor includes a feature to help you balance punctuation automatically. When you type a closing brace, bracket, or parenthesis, CodeWarrior looks for the matching opening punctuation to make sure it is balanced. If the editor doesn't find a match, it beeps so you know something is wrong right away. This lets you find the problem while you are right in that particular section of code, and the context is fresh in your mind.

However, even if the editor finds a match, that match might be the wrong one! So every time it finds a match, the editor highlights the corresponding opening punctuation mark. In this way you can see where the opening mark is, and decide if it is the correct mark. If the opening mark is not visible in the window, CodeWarrior automatically scrolls to it for a brief moment, and then scrolls back to where you were.

This whole process, like dynamic scrolling, can be a little disconcerting the first few times you see it happen. Nevertheless, it is a wonderful feature that can save you a tremendous amount of time later on tracking down punctuation problems. Before long, you will find yourself unconsciously listening for the beep that alerts you to a balancing problem every time you hit a close parenthesis.

Like many other cool features, the check for punctuation balance is on by default in CodeWarrior, but you can turn it off. You control this feature through the editor's Editor Settings preferences panel, as shown in Figure 4.5. You click the Balance While Typing check box to toggle the state on or off.

You also control how long CodeWarrior displays the balancing punctuation. You set the value in the Flashing Delay text field in this preferences panel. The acceptable range is 0 to 240, and the number represents units of 1/60th of a second. Even if the value is zero, CodeWarrior still checks for the balancing punctuation, scrolls to it ever so briefly, and beeps if there is none.

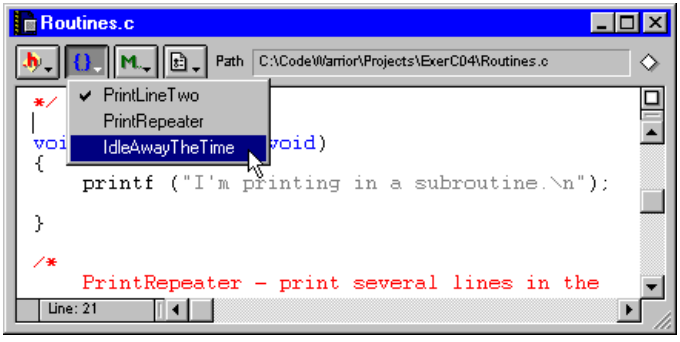
**TIP**

If you double-click a brace, bracket, or parenthesis, the editor selects all the text between it and its balancing mark. This is a great way to select all the code inside braces prior to a copy and paste operation, or to shift the block left or right. It also gives you a really good way to see exactly what's inside the punctuation. For example, you can double-click the parentheses inside nested statements of any kind, so you can be sure everything is set up properly.

## Jumping to Any Routine

Another task you are likely to perform on a regular basis is to go to a particular routine in a file. You know the name of the routine, but how do you find it? Well, you can scroll around looking for the routine, or rely on your memory about where the routine is located in the file, but there's a better way. Just select the routine from the Routine pop-up menu, illustrated in Figure 4.12.

FIGURE 4.12
The Routine pop-up menu




All the routine names in a file are listed in the order in which they appear in the file. You can see them in alphabetical order if you press the Alt key and click the menu.

If you prefer to always have the routine names in this pop-up menu listed alphabetically, you can do that as well. You control sorting order

**TIP**

In a Pascal file, procedure names are in italics, function names are in plain face, and the main program is in bold.

in the Editor Settings preferences panel shown in Figure 4.5. Click the Sort Function Popup check box to turn this feature on or off. By default, this option is off.

If you turn this option on, the routine names appear alphabetically when you examine this menu. (If this option is on, the Alt key puts the routine names back in appearance order).

**TIP**

If you hold down the Shift key while choosing a function in the pop-up menu, CodeWarrior selects all the code in the function. This shortcut is very useful if you're planning to move the code elsewhere.

## Setting Markers

You know that sometimes you have special places in your code that you return to over and over again. It might be a critical routine, for example, or something that you want to refer to regularly. A good editor should let you identify these places so you can go to them easily.

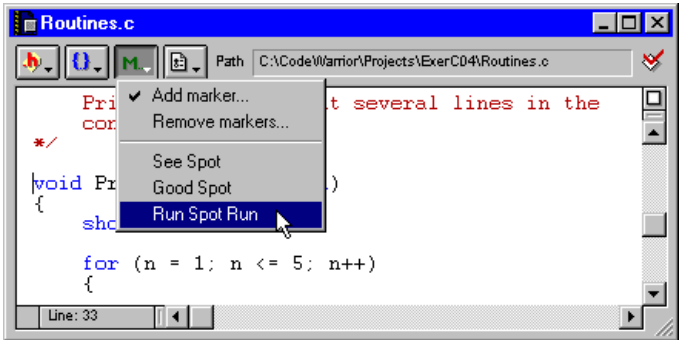
In CodeWarrior, you can set markers at any location in any file. The process is simple. The controls are in the Marker pop-up menu, as shown in Figure 4.13.

To add a marker, first set the insertion point where you want the marker to be in the file. Then choose the Add marker item from the Marker pop-up menu. When you do, a dialog box appears that asks you for a name for the marker.

After you add one or more markers to a file, you can jump to any marker by simply selecting it from the Marker menu. You jump

FIGURE 4.13

The Marker pop-up menu lists every marker in a file, and lets you add or remove markers.

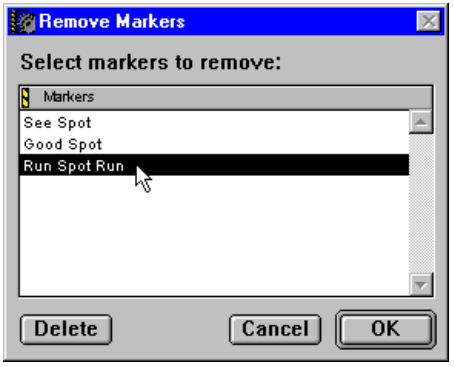



instantly to that location in the file, and the insertion point appears at the mark.

To remove a marker, choose the Remove markers item from the pop-up menu. A different dialog box appears that lists every marker in the file, as shown in Figure 4.14. Select those you want to remove, then click the Delete button, or press the Backspace or Delete key.

FIGURE 4.14

Use the Remove Markers dialog box to remove a marker from a file.



**TIP**

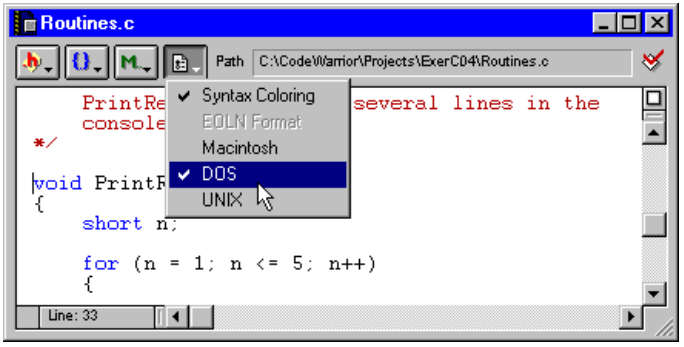
C/C++ programmers have another way to mark a file. Anywhere in your source file, you can add a line of code like this: **#pragma mark markerName**. Markers created with **#pragma mark** show up in the Routine pop-up menu, not the Marker menu.

## Controlling Line Endings

CodeWarrior is designed to support code for multiple targets and to run on multiple hosts. As such, the engineers who built CodeWarrior are very aware of cross-platform issues. One such issue is how different platforms mark the end of a line. You control which platform the file is designed for with the Options pop-up menu shown in Figure 4.15.

FIGURE 4.15

The Options pop-up menu lets you control the format CodeWarrior uses to save files.




The dimmed EOLN Format item is simply a header indicating that the items below it control end-of-line format. When you save the file, CodeWarrior uses the appropriate line ending format for the platform you choose. The check mark indicates the current status of the file.

**TIP**

This feature is a great way to convert source code files from one platform to another.

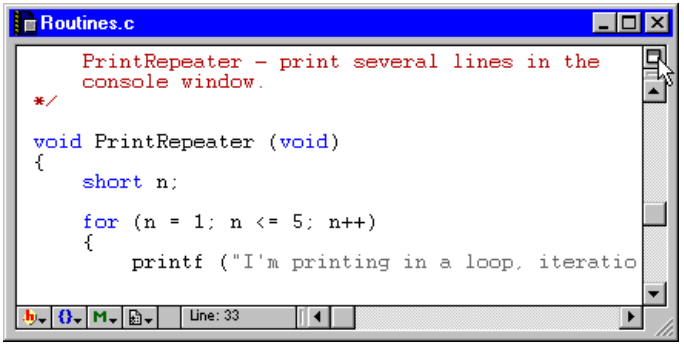
## Controlling the Look of the Editor

As you have seen in many illustrations in this chapter, large icons line the top of the CodeWarrior editor window. These icons give you access to a series of four important pop-up menus—interface, routine, marker, and options. However, the CodeWarrior engineers know that your screen real estate is vital. You might prefer to have the space available for a few more lines of code rather than devote it to large-icon, 3D, user interface elements.

By now you should be accustomed to the extent that CodeWarrior lets you be in control. You can hide these icons, and replace them with much smaller buttons that fit along the bottom of the window. Simply click the little pop-up menu display button at the top of the vertical scroll bar, above the splitter control. This button toggles between a) showing the menus as large 3D icons, and b) displaying them as small icons at the bottom of the window, to the left of the horizontal scroll bar as shown in Figure 4.16.

FIGURE 4.16

You can control the appearance of the editor window in various ways, and use small icons instead of large buttons for the pop-up menus.




The arrangement in Figure 4.16 gives you more room in the window to display code. The menus are still available—they just have smaller buttons.

**TIP**

You can make all your source code windows use this feature by default. First, set the active window to use small icons for menus. Then choose the Save Default Window item from the Window menu. CodeWarrior remembers that this is how you want your source code window to look. The next time you open any source code window, it uses the small icons.

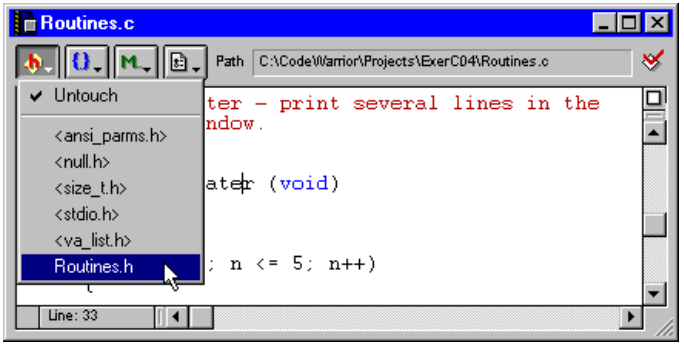
## Opening Related Files

Files are frequently interrelated, which means that you often need to see text not just in one file, but in two or more related files. This need is especially true of interface files included in a source file. This is such a common need that CodeWarrior offers several mechanisms to help you open included interface files quickly and easily, no matter where they are.

As you know, you can get to the interface files quickly through the File pop-up menu to the right of the file name in the project window. The editor provides the same functionality in the Interface pop-up menu, usually found at the top left of any editor window. Figure 4.17 shows this menu.

FIGURE 4.17

The editor's Interface pop-up menu lists all the include files for a source file.



If you are looking at the file name in the source code itself, for example, in a **#include** statement, you can select the file name and choose the Open Selection command from the File menu (Ctrl+D). CodeWarrior opens the file for you. When you encounter a file name in code, you can quickly double-click and press Ctrl+D to open the file. This is a favorite among CodeWarrior programmers. You don't have to select the file name extension; the editor does it for you automatically. Because interface files are usually included at the start of a source file, you can scroll to the top of the file, select the name of the file you're interested in, and open it instantly with Ctrl+D.

**TIP**

You can use the interface pop-up menu to touch or untouch a file for the compiler, just as in the project window.

## Working with Multiple Windows

The entire CodeWarrior MDI workspace is available to you for editor windows. You can open as many windows as your system's memory allows. However, a plethora of windows can be very confusing. CodeWarrior does help you sort through windows efficiently, however.

Of course, you can simply click any part of a visible window to make it active. This brings it to the foreground and conceals anything behind the window. However, several commands help you manage windows more effectively than hunting and clicking. You will find each of these commands in the Window menu, as follows:

- **Tile Vertical:** resizes windows so they appear side by side
- **Tile:** resizes windows so they appear above and below each other
- **Stack:** creates a staggered cascade of windows
- **Zoom:** toggles between full size and previous size

The Tile Vertical and Tile commands work best if you have only two or three editor windows open. The Stack command is useful when you have many windows open, and you want to see all the title bars.

The Zoom command also might need a little explanation. This command is not the same as clicking either the minimize or maximize buttons, or changing magnification. The editor has a default "full size" for a window. If the window isn't already at that size and you Zoom the window, it resizes to the full size. If you Zoom again, the window returns to its previous dimensions. The Zoom command toggles between these two states, full size and whatever size the window was previously. In combination with the standard minimize and maximize controls, this command gives you a whole set of options you can use to put your windows at a good size at any moment.

The Window menu also lists every open window. You can switch among windows by choosing the window name from the menu. You'll learn more about closing windows a little later in this chapter.

**TIP**

You can cycle through all the open windows one after the other with Ctrl+Tab.

## Saving a File

When you create a new file, you must save the file on disk or it will be lost when CodeWarrior quits. Similarly, if you make changes to an existing file, you must save your changes to disk or they will be lost. You can save files individually. You can also save files automatically in certain circumstances.

CodeWarrior also gives you complete control over the names and locations of your files. This section discusses each of these topics in detail.

## Saving One File

To save a single file simply choose the Save command in the File menu. CodeWarrior saves the active document to disk. The Save All command in the File menu saves all open documents to disk.

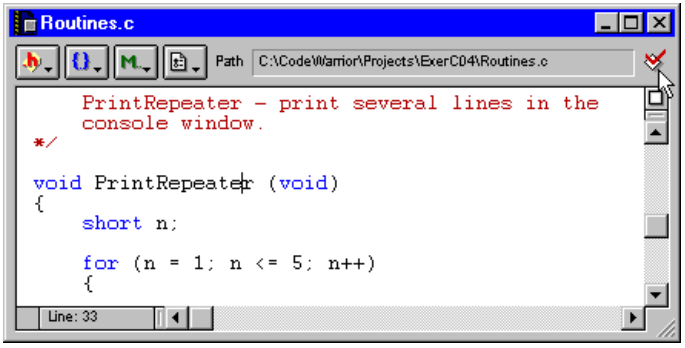
If the file is new and has never been saved, CodeWarrior displays the standard dialog box asking you to name the file. You can also specify the folder in which you want the file to reside.

**TIP**

You can always tell when a file has been changed and needs saving. Look at the diamond to the right of the file path display near the top of the editor window. If you see a red check mark, the file has been changed and needs to be saved, as shown in Figure 4.18. When you save a file, the check mark disappears.

FIGURE 4.18

The check mark tells you a file has changed and has not been saved.




## Saving Files Automatically

You have seen the endless-loop instructions on shampoo bottles: lather, rinse, repeat. Sometimes it seems that software development is

a similar endless loop: write code, build, debug, repeat. Often you rush through this cycle as quickly as possible. CodeWarrior is a great tool, because it helps keep the cycle as short as possible.

However, in the rush through the development cycle it is easy to overlook a critical safety feature. After you write or modify code, you should save your work before you compile and run it. Why? Because your code may crash big-time, taking the entire machine along with it. If you have not saved your work, you're out of luck.

The engineers who build CodeWarrior face the same problems. So they built into CodeWarrior a mechanism to automatically save all files before you compile. This feature is on by default, but you can turn it off and on at will. It is a global preference for the source code editor.

You control this feature through the Editor Settings preferences panel, as shown in Figure 4.5. Click the Save All Before "Update" check box to toggle the setting on or off. When this option is on, CodeWarrior automatically saves all changed files when you update your project.

**TIP**

The master knows when to delegate tasks to an assistant. Although you can turn this feature off, *don't*. It has saved me more than once, and I guarantee it will save you. It is simply too easy to forget to save files before testing the code. Let CodeWarrior do the work for you.

## Renaming and Saving a File

To save a file using a new name, use the Save As command in the File menu. When you do, CodeWarrior displays the standard dialog box asking you to specify the new name and directory. If the file you are renaming is in the current project, CodeWarrior replaces the old name with the new name in the project window. The original file is now disconnected from the project.

If you don't want this to happen—that is, if you want to save a copy of the file but not replace the file in the project—use the Save A Copy As command in the File menu. This works exactly like the Save As command, but CodeWarrior does not replace the project file with the copy you create.

## Reverting a File

Sometimes you start making changes to a file, and you realize you started in the wrong direction. What you'd like to do is throw away the changes and start all over by returning to the condition the file was in the last time you saved it. In such a case, choose the Revert command from the File menu. CodeWarrior replaces the contents of your window with whatever was in the file the last time you saved it. This effectively throws away any changes you made since the last time you saved the file, and lets you start fresh.

**WARNING** Reverting a file cannot be undone, so be careful. CodeWarrior will ask you to confirm the command, just to be sure.

## Saving Window Information

CodeWarrior lets you save information about the window along with the contents of the window. You can save:

- Font preferences
- Window position and size
- Selection position

The font preferences are the font and size used for text in the current window. If you save this information, the window displays in its own font and size, regardless of the global font preferences you have set in the editor's Font & Tab preferences panel.

If you save the window position and size, the window reappears at its prior location when you reopen the file. This is useful if you have tiled windows or located them carefully for viewing multiple windows simultaneously.

CodeWarrior can also remember where the insertion point was in the file, and any selection range. When you reopen the window, the insertion point is right where you left it, and any text that was selected remains selected.

You control which of these features are saved with the window in the editor's Editor Settings preferences panel, as shown in Figure 4.5.

In addition to saving the preceding information, you can also choose the Save Default Window command in the Window menu. This command saves the window's internal arrangement—how the pop-up menus appear, either as large buttons at the top or small icons at the bottom of the window. It does not save the font, size, and location preferences. That happens automatically if you choose those options in the Editor Settings preferences panel.

## Closing a File

Every open file has an associated window. When you open a file, it gets its own window. When you close the window, you close the file. You can close a window using the Close command in the File menu, or click the close box. If the file has unsaved changes, CodeWarrior asks you if you want to save changes.

**TIP**

To close all editor windows at once, choose the Close All item in the File menu, or press the Alt key and click in the active window's close box. CodeWarrior closes all windows automatically, asking you whether to save any changed files along the way.

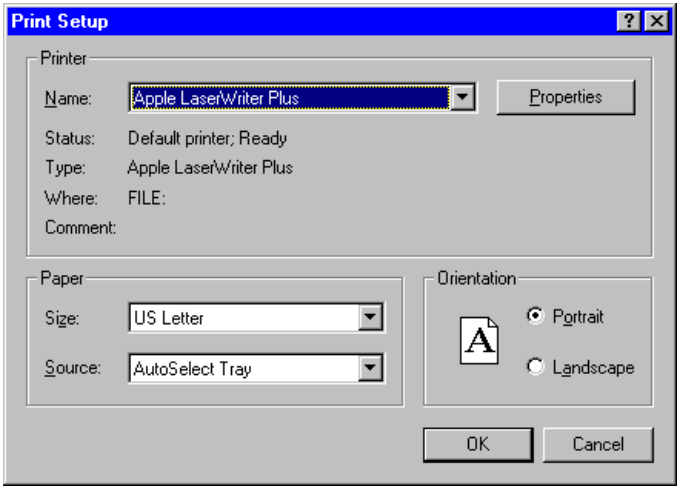
# Printing a File

Sometimes you want to print a hard copy of your code. You can print the contents of any window in CodeWarrior, not just code. You can set a variety of print options before printing. If you don't set the options, CodeWarrior uses default values.

## Setting Print Options

To tell CodeWarrior how you want to print a file, choose the Print Setup command in the File menu to see the standard Print Setup dialog box, as shown in Figure 4.19.

FIGURE 4.19
The Print Setup dialog box lets you set all the options available for your printer.




You can pick a printer, select paper size, paper source, and orientation. Depending on your printer, when you click the Properties button you may be able to set many more features, such as the number of copies.

### Printing a Window

Make sure the window you want to print is the active window and then choose the Print command in the File menu. CodeWarrior sends your file to the printer you chose in the Print Setup dialog box.

## Hands-On Exercises

In this exercise you're going to work with the editor, and you'll explore all the important features described in this chapter. You'll create new windows, open existing files, split panes, work with each of the controls in the editor window, and more.

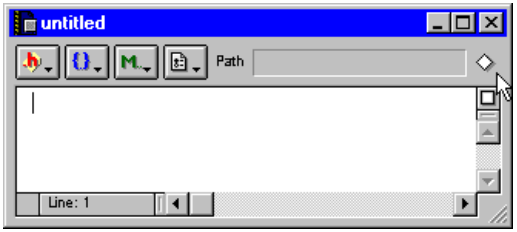
**1. Open a project and create an editor window.** If CodeWarrior is not running, launch CodeWarrior and then open the Greeting.cwp project file in the exercise folder for this chapter, Exer04. After you have the project open, choose the New item (Ctrl+N) from the File menu. When you do so, an empty, untitled editor window appears.

**2. Play with text.** In this step you just type in some text to familiarize yourself with cursor control and text selection in CodeWarrior. Before you start typing, however, notice the diamond icon at the top right of the window, as shown in Figure 4.20. Now that you've had a look, start typing and take another look at the diamond—you'll see it changes to a check mark as soon as you change the contents of the window. The check mark tells you the file has unsaved changes.

Type in some more text. You don't have to write the opening lines of code to the greatest game since *Marathon* or *Doom*. You don't even have to write any code at all. The point is simply to explore the editor. Enter a few lines of text or code, and then start to play around with it.

For example, after you have typed something, use the Backspace and/or Delete keys to see how they work. You might also write a lot of text on one line, and notice that the line doesn't break when you

FIGURE 4.20
The plain diamond changes to a check mark when you make a change to the file.




reach the right edge of the window. You should also explore cursor control and how to select text. Experiment for a few moments and study how the insertion point moves and the selection range changes by:

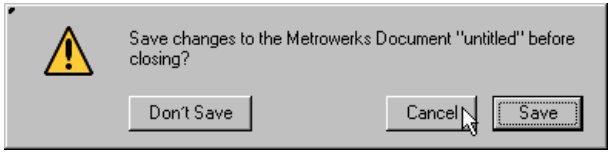
- Using Ctrl+arrow keys
- Clicking with the mouse
- Double-clicking a word
- Triple-clicking a line
- Using Shift+click to select a range
- Using Shift+arrow keys
- Using Shift+Ctrl+arrow keys to select text

These are all fundamental operations you are probably already familiar with (except perhaps the Ctrl key effects), so the preceding explorations should only take a moment or two. When you're comfortable, continue with the next step.

**3. Save a file.** Because you have made changes to your new file, you cannot close it without being asked to save it. Try it now—click in the close box, or choose the Close item from the File menu. The dialog shown in Figure 4.21 appears. Click the Cancel button, because I want to show you a couple of other features before you close this file.

Choose the Save item (Ctrl+S) from the File menu to see the standard file saving dialog. Give the file a name (end it with an extension

FIGURE 4.21
CodeWarrior warns you if you attempt to close a changed file without saving changes.




such as .c or .txt so CodeWarrior will recognize it if you want to open it again), locate where you want to put it, and save the file. The window remains open, but notice that the check mark icon at the top right of the window has changed into a plain diamond again. If you make any subsequent change, the check mark reappears so you know the status of the document on display in the window.

**4. Undo and redo changes.** In this step you explore the undo and redo features of the editor. To make sure you see how it all works, follow the instructions in this step closely.

Change the contents of the window in precisely this order. First, select a word and cut it from the file (don't use the Backspace key to delete it, use the Cut command from the Edit menu). Then, move the insertion point to some other place in the file and type some new text. What you have done is set up a chain of two actions—one cut operation, and one typing action.

Now look in the Edit menu at the Undo command, and you'll see that you can undo the most recent typing action. That item reads Undo Typing. Choose Undo, and your just-typed text disappears.

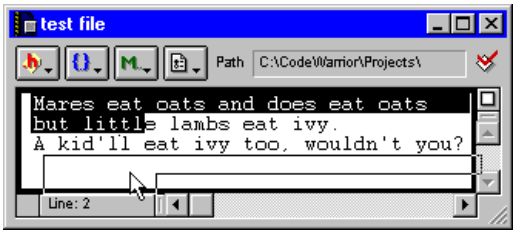
Look at the Undo item again, and this time it says Undo Cut. Choose Undo again, and your cut text reappears.

You can also Redo tasks after you Undo them. Choose the Redo Cut item and your cut text disappears again. Choose Redo Typing and your typed text reappears. Get familiar with the keyboard accelerators for these commands—they are very useful. Use Ctrl+Z to undo and Shift+Ctrl+Z to redo.

**5. Move and copy text with drag-and-drop.** Drag-and-drop is a great way to move text around inside a window, or between CodeWarrior windows. First, select a range of text. Then click and drag the selected text to some other location in the file. Figure 4.22 shows what it looks like—a frame indicates the bounds of the original text, and a flashing cursor indicates the insertion point for the new text. Just drop the text where you want it to be, and it disappears from its original location and reappears at the new spot.

FIGURE 4.22

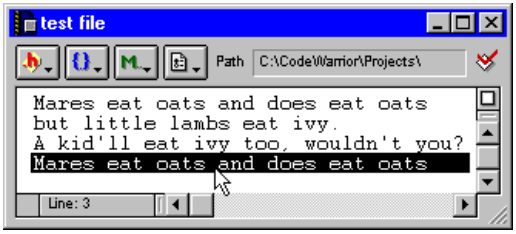
Dragging is a quick way to relocate a range of selected text.




You can also use this technique to copy text inside a window. Just hold down the Alt key as you drag the text. When you drop it, the original text remains where it is, but a copy appears where you dropped it. In Figure 4.23, the top line of text has been duplicated in this fashion, and appears again at the bottom of the text.

FIGURE 4.23

You can use drag-and-drop to duplicate text.

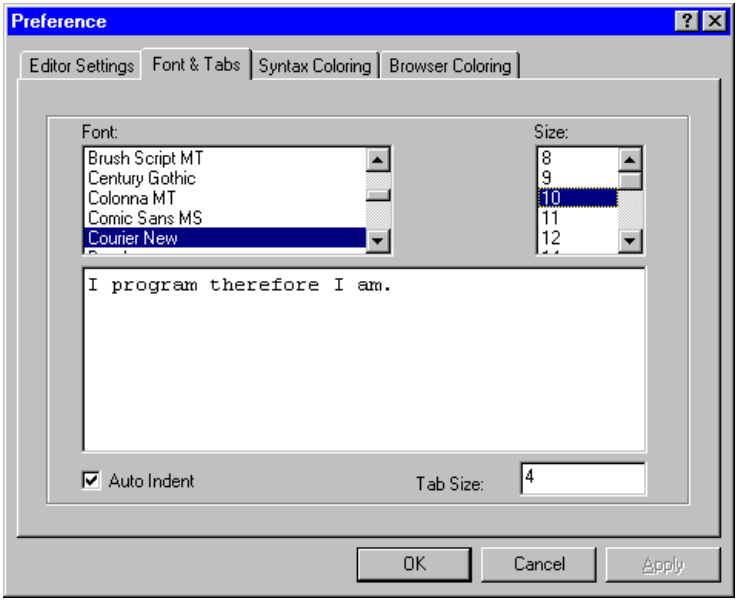

test file
Path
C:\CodeWarrior\Projects\
Mares eat oats and does eat oats
but little lambs eat ivy.
A kid'll eat ivy too, wouldn't you?
Mares eat oats and does eat oats
Line: 3

You can drag text between windows as well. If you do, the original text remains where it is. Any drag across windows is considered a duplication or copy operation, not a move. There's no need to press the Alt key. Create another window if you want and give it a try. You can undo and redo drag operations as well!

**6. Set text options.** In this step you learn how to set the font and type size used in editor windows, how to control color, and modify various editor settings. To begin, choose the Preferences item from the Edit menu to open the global preferences, and then display the Font & Tabs panel, shown in Figure 4.24. Play around with the settings here—pick a different font or change the size—until they suit you.

FIGURE 4.24

Choose font and size for all your editing windows.




While you're here, you should also explore the Syntax Coloring and Editor Settings panels. Once again, play with the settings until you like the arrangement, and click OK to accept all changes to all panels.

Then observe and explore in the editor window to see the effect of your changes.

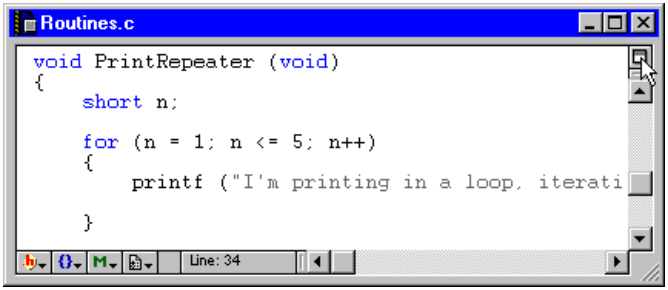
For example, you might turn off drag-and-drop editing in the Editor Settings panel, and then try to drag some text around. Change the color of comments, and add some comments to your file. Feel free to modify and explore as you see fit. When you're all done, close this window. Save the changes if you want, or not. You won't be using it again in this exercise.

In this first part of the exercise you have successfully created a window, modified its contents, and saved a file on disk that contains the contents you placed in the window. You have also explored some of the basic features of the editor. That's a great start, but there's a good deal more. For the rest of this exercise you'll use existing source files so you can learn in a controlled fashion all about the more advanced features of the editor. Let's start by opening a file.

**7. Open an existing file and modify its appearance.** The project window gives you easy access to your files. Double-click the Routines.c file in the project window to open that file. Notice that the pop-up menus are large and cover the top part of the window area. Click the control at the top of the vertical scroll bar to move these menus down to the bottom of the window, as shown in Figure 4.25.

FIGURE 4.25

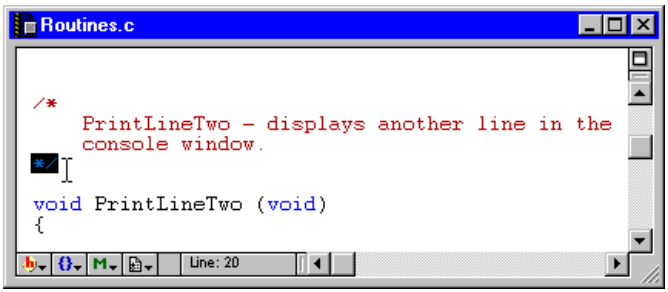
Small icons at the bottom of the window give you more room for text.



Decide which way you like the window to be, then choose the Save Default Window item from the Window menu. All editor windows will appear in this configuration. Go ahead, try it. Create a new window, or open the Main.c file, and see what it looks like.

**8. Observe the effect of coloring.** As you look at the contents of the window, you'll notice that the text appears in color. If you modified the color settings back in Step 6, the current file will reflect those changes. As a good example of the effect color can have, locate the comment just above the **PrintLineTwo()** function, and delete the terminating "*/" text at the end of the comment. Figure 4.26 shows this spot in the file.

FIGURE 4.26
Delete the terminating comment at this routine.

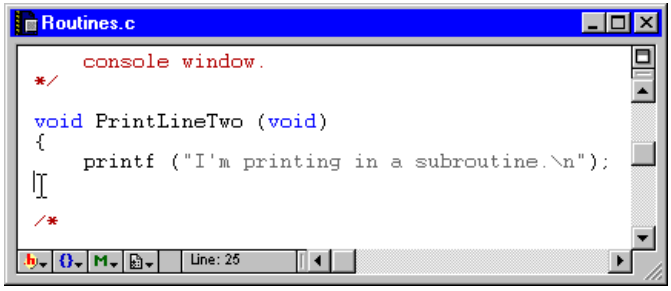



After you delete the terminating */, observe what happens to the code right beneath the comment. It all changes color, because it is now part of the comment. After you have seen the effect, restore the file to normal with the Undo command.

**9. Check for balanced punctuation.** In the Routines.c file, locate the **PrintLineTwo()** function. To test how punctuation balancing works, first delete the closing brace at the end of the function, as illustrated in Figure 4.27. Now, keep an eye on the brace at the beginning of the function while you retype the closing brace where it belongs. The opening brace highlights for a moment because it is the balancing punctuation mark.

FIGURE 4.27

Delete the closing brace at this routine.



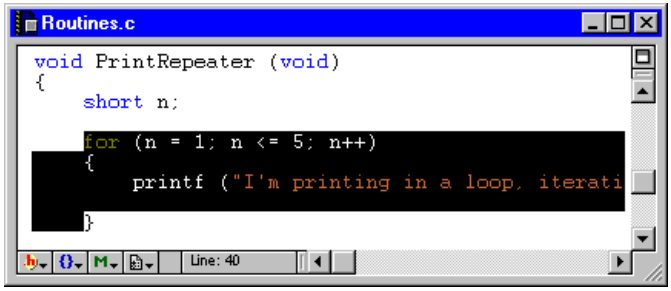

Now type a second closing brace—an error—to hear what happens. Because it can find no matching brace, the computer beeps to alert you to a balancing problem.

Finally, you can do one more trick with balancing. Double-click a brace or parenthesis somewhere in this function, and all the code inside that punctuation becomes selected. This is a really handy way to select a useful range of text.

**10. Shift code left and right.** Another nice feature of the editor is its ability to nudge sections of code left and right so you can control indentation. Figure 4.28 shows a range of selected text in the **PrintRepeater()** function. Locate this function in the file, and select this text.

FIGURE 4.28

Select a range of text to prepare for shifting.



Now choose the Shift Left (Ctrl+[) item from the Edit menu, and observe what happens to the selected lines. Use the Shift Right (Ctrl+]) command as well. You can use these commands to align text properly.

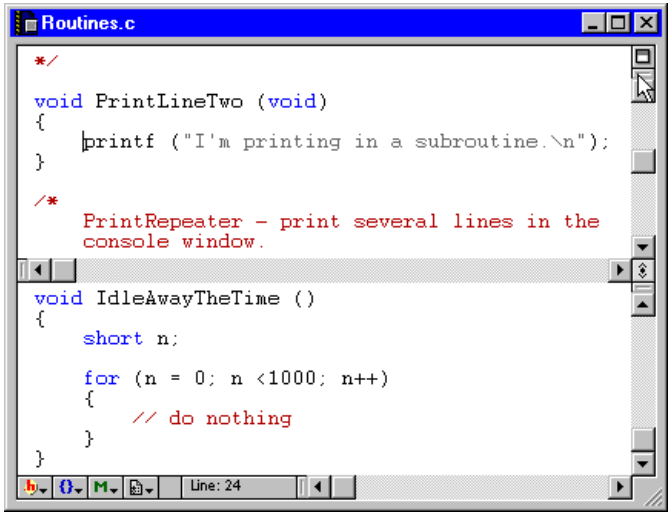
**TIP**

When you type, the editor automatically indents a new line to the left edge of the previous line. The Editor Settings preference panel lets you turn this auto-indent feature off if you wish.

**11. Create and destroy split panes.** Double-click the pane splitter at the top of the vertical scroll bar to split the Routines.c window into two equal horizontal planes. Figure 4.29 shows the result.

FIGURE 4.29

Multiple panes make it easy to see different parts of the same file.




Use the pane resize box—that's the area between the two vertical scroll bars—to resize the panes. Play around with the multiple panes until you feel comfortable. For example, change the insertion point from one to the other, and then use the Home, End, Page Up, or Page

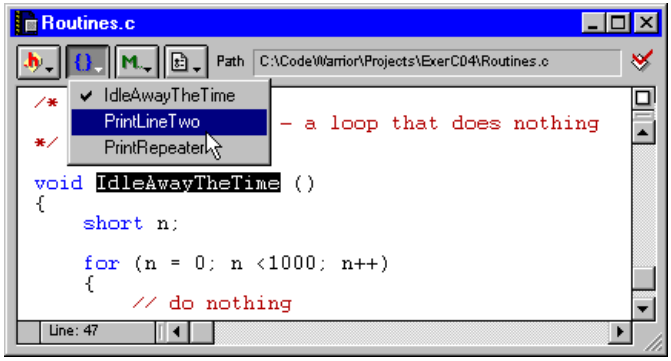
Down keys to scroll around the pane. You can also copy and paste text, or drag text between panes. Give it a try.

If you drag text, notice a subtlety in the way dragging behaves. Although you are dragging text from one pane to another, because the panes are really part of the same window this is considered a "move" operation and the text disappears from the original location when you drop it. To duplicate text across split panes, press the Alt key while dragging.

**12. Use the Routine pop-up menu.** Click on the routine pop-up menu to see a list of functions in Routines.c, and note that the function with the insertion point has a check mark to the left of the name. Choose any function and you'll jump to it immediately. Press the Alt key as you display the menu, and you'll see the names in alphabetical order as shown in Figure 4.30.

FIGURE 4.30

You can see the functions in a file in alphabetical order in the routine pop-up menu.



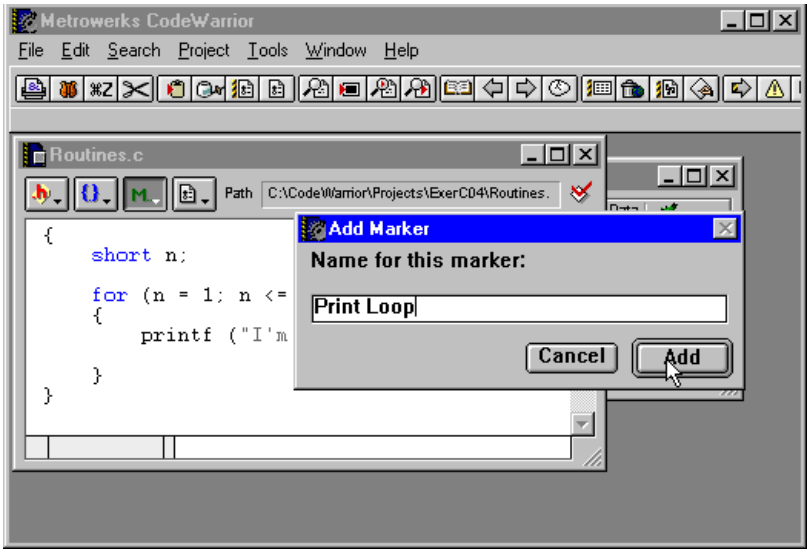

**TIP**

Don't forget, you can use the Editor Settings preference panel to make alphabetical order the default condition if you wish.

**13. Set a marker and jump to it.** In this step you work with the editor's ability to set markers anywhere. With Routines.c as your active window, make sure you notice where the insertion point is currently located. It really doesn't matter where it is, but feel free to put it someplace interesting, such as at the end of a function. In Figure 4.31, it is located inside the printing loop in the **PrintRepeater()** function. Then use the Marker pop-up menu and choose the Add Marker item. In the resulting dialog box, shown in Figure 4.31, enter a name for this location, and click the Add button to add the marker to the file and close the dialog box.

FIGURE 4.31

You can add a marker anywhere in the file, and name the marker anything you want.



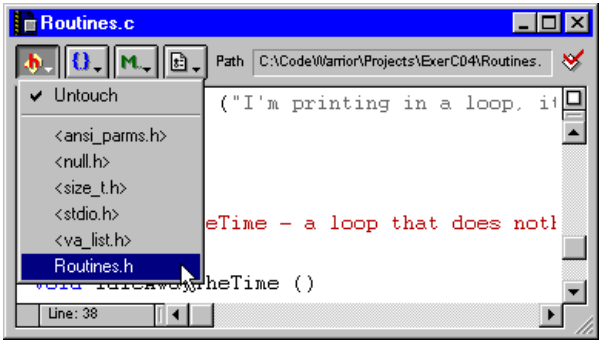

Back in the Routines.c file, move the insertion point elsewhere. Then look in the Marker pop-up menu and you'll see your new mark listed. Choose that item, and watch what happens inside the Routines.c window—the text scrolls automatically to the location of your marker.

You can add as many markers as you'd like. Markers are attached to the file, so the list in the Marker pop-up menu reflects markers available in the current file only.

**14. Open an included file.** In this step you use the editor's ability to locate included files automatically. For this step to work, the project has to be compiled to give the compiler an opportunity to read the source files and see what files are included.

To begin, use the Bring Up To Date command in the Project menu to compile all the code in the project. When the compile is finished, the project manager knows which source files include which interface files. Display the Interface pop-up menu for Routines.c, and choose the Routines.h file from the list of files, as shown in Figure 4.32. This file opens, and you can study the prototypes for the functions in Routines.c.

FIGURE 4.32
Open included files with the Interface pop-up menu.




**15. Work with windows.** This is the last step for this exercise. In this step you use CodeWarrior's window-management features to activate, tile, print, and close windows.

You can cycle through all the open CodeWarrior windows with the Ctrl+Tab accelerator. Use it now to activate each open window in turn. Each time you press the keys, a new window comes to the front. This is analogous to Windows' Alt+Tab feature for cycling through applications and desktop windows.

You can tile editor windows horizontally or vertically. If you have been following these directions closely, you should have two editor

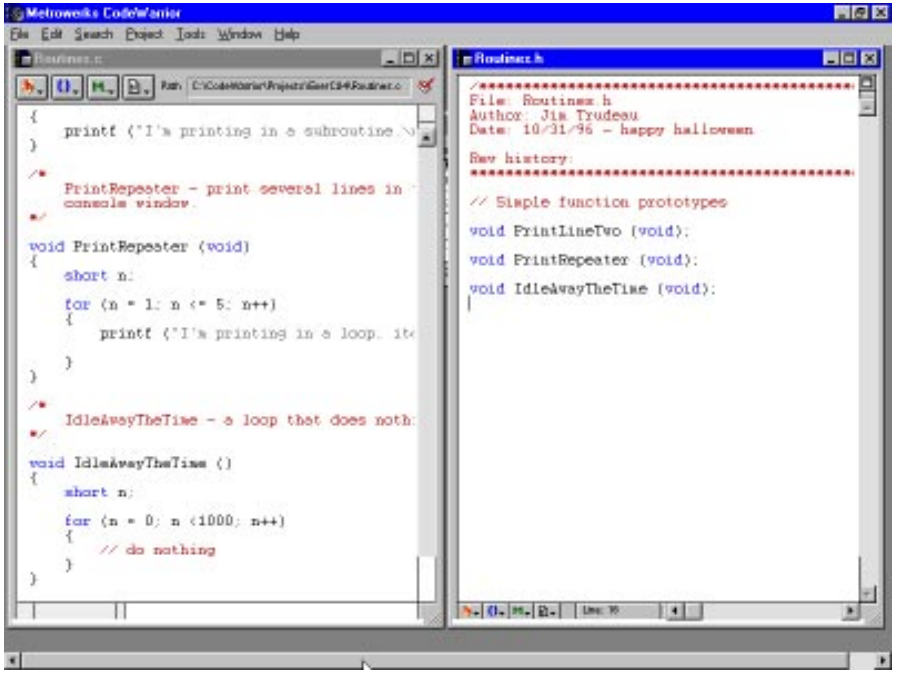
windows open—Routines.h and Routines.c. If you have more editor windows, close them now. You can leave the project window open; tiling does not affect the project window. When you have just the two editor windows open, choose the Tile Vertical item from the Window menu. In response, CodeWarrior resizes the windows as shown in Figure 4.33. Experiment with the Tile command as well. You might want to open a third or fourth window, and see how they tile.

If you want, while you have a window open, print its contents. Printing works like it does for any Windows application, so you should be comfortable with how it works in CodeWarrior.

That's it for this exercise, so let's close down. To close all the editor windows at once, Alt+click in the close box of any editor window (or choose Close All from the File menu). If any changes remain unsaved, you'll be asked whether to save them or not. Once again, like tiling,

FIGURE 4.33

Tiling windows vertically lets you see files side by side.


Metrowerks CodeWarrior
File Edit Search Project Tools Window Help
Routines.c
Routines.h
printf ("I'm printing in a subroutine.\n");
PrintRepeater - print several lines in the console window.
void PrintRepeater (void)
short n;
for (n = 1; n <= 5; n++)
printf ("I'm printing in a loop. it
IdleAwayTheTime - a loop that does noth
void IdleAwayTheTime ()
short n;
for (n = 0; n <1000; n++)
// do nothing
File: Routines.h
Author: Jim Trudeau
Date: 10/31/96 - happy halloween
Rev history:
// Simple function prototypes
void PrintLineTwo (void);
void PrintRepeater (void);
void IdleAwayTheTime (void);

closing all windows does not affect the project window because you are most likely to want keep the project open.

You can exit CodeWarrior at this time if you wish—any changes to the project are saved automatically.

## Summary

This chapter took you deep inside the CodeWarrior editor. You learned about file management, code authoring, editing, and navigation. You know how to work with source code files directly. You can create, open, save, close, revert, and print your files.

The editor is now familiar stomping grounds. You can work with multiple windows, you know how to split panes in the editor, modify color usage, and save the default appearance of a window. You also got to know the various controls in the editor window.

More importantly, you learned how to use the editor to accomplish your goal: writing code quickly and efficiently. You can add, move, delete, copy, paste, and drag code around. You can undo and redo actions. You know how to balance punctuation. Best of all, you learned powerful navigation features that help you move around in a file. You can jump to any function, any included file, or any marker with a simple click of the mouse.

That's real power. But more discoveries await you as you continue your exploration of CodeWarrior. In the next chapter you'll learn about the editor's powerful search and replace features. After that, I'll take you through the CodeWarrior browser—a code navigator without peer. The adventure has hardly begun!

CHAPTER

FIVE

5

# Searching and Replacing Your Code

- Understanding the Search Tools
- Finding Selected Text
- Finding in a Single File
- Finding in Multiple Files
- Cool Features
- Hands-On Exercises

This chapter explores the editor's powerful search and replace features. You know you have to hunt for particular elements in your code on a regular basis. The ability to search for snippets of text is vital to you as a programmer.

Sometimes you just want to find a particular occurrence of a word, because you know that's where you want to be in your code. That's usually a pretty straightforward task to accomplish in any text editor. However, at times your goal is more demanding. For example, you might decide to change the name of a global variable to something more descriptive. When you make this kind of decision, you are faced with the task of finding each and every use of that variable name anywhere in your code and then changing it.

This can be a daunting task to say the least, particularly in a big project. In fact, the difficulty such a chore entails can lead you to avoid making changes, even when it is in your long-term best interest to do so.

CodeWarrior gives you the tools you need to make these kinds of changes easily—even automatically if you wish. You can use these tools to look for and replace text in one file, or in a collection of files.

We'll follow a familiar path. First we'll explore the search and replace interface, just so you know what the tools look like. Then we'll analyze the common tasks you want to perform to see how you can use the editor's search and replace tools to best effect. We'll start with simple tasks and move on to increasingly complex operations and cool features. By the time you finish this chapter, you will be a master of search and replace in CodeWarrior.

The editor's search and replace features strike a very good balance between simplicity and complexity. As you'll see in this chapter, CodeWarrior has very simple and easy ways to accomplish common and ordinary tasks, but the editor doesn't stop there.

Sometimes you need powerful tools to perform intricate tasks. While it takes more effort to master the complex aspects of the tool set, CodeWarrior gives you the real power you need no matter how complex your job.

# Understanding the Search Tools

Let's take a quick look at the search and replace design decisions made by the CodeWarrior engineers. These are the behind-the-scenes principles that control the way the tools work. You'll see these principles reflected in the interface, and in how you use the tools. If you know why the tools work the way they do, you'll understand how to use them better.

## Understanding the Search Design

Search and replace is a text-related function, so it ties in closely to the source code editor. When searching and replacing, the *find* string is the text you are looking for. I'll call this the *target text* to avoid confusion with use of the word "find" in a variety of different contexts throughout this chapter. The *replacement text* is the text you want to use (if any) to replace the target text.

When you first launch CodeWarrior, these two strings of text—the target text and the replacement text—are empty. You cannot begin a search unless you put some text in the target text. However, searching is possible even when the replacement text is empty.

If you do a replace operation, whatever is in the replacement text replaces the found target text. If the replacement text is empty, the found target text is simply deleted (that is, replaced with nothing). Table 5.1 summarizes what you can do depending upon whether the target text and replacement text have been filled.

Once you fill either the target or the replacement text, the editor remembers the string until you change it.

TABLE 5.1 Searching requires target text

| Target Text | Replacement Text | You Can |
|---|---|---|
| empty | empty or filled | not search |
| filled | empty | search and delete |
| filled | filled | search and replace |

In addition to the target and replacement text, the editor has a variety of features you can use to control the direction, range, and scope of the search. Direction means you can search forward or backward in a file. Range means what file, part of a file, or group of files you search in. Scope means how closely the actual text in the file matches the target text you are looking for.

With the preceding in mind, let's now look at the controls you use to manage search and replace.

## Understanding the Search Controls

CodeWarrior's search and replace commands are all located in the Search menu, as shown in Figure 5.1. Think of this menu as "search central." This menu's commands are the focus of search and replace functionality. In this chapter you'll become familiar with all the search commands.

The simple way to begin a search in CodeWarrior is to choose the Find command from the Search menu (Ctrl+F) to display the Find window shown in Figure 5.2.

**TIP**

If you want to be a real power-user, you'll learn the Search keyboard equivalents as well. You can control most aspects of a search right from the keyboard.

FIGURE 5.1

Commands in the Search menu control the CodeWarrior search engine.

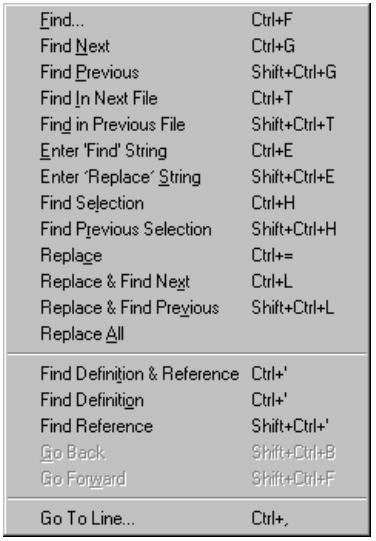
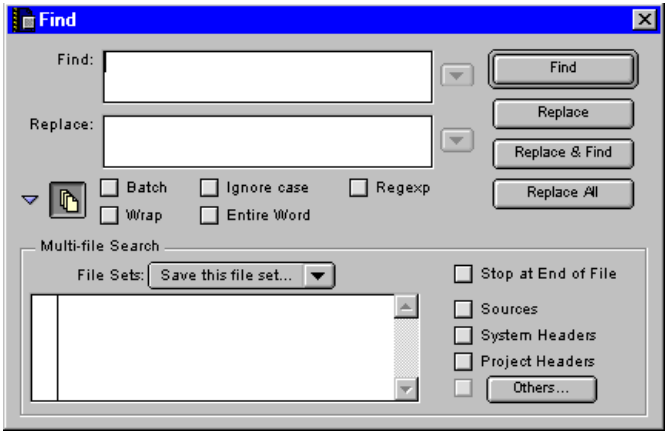



FIGURE 5.2

The Find window contains controls for specifying all search parameters.




A brief glance at Figure 5.2 shows you many displays, buttons, pop-up menus, check boxes, and so forth. Using these controls is actually a lot easier than it may look. As you go through this chapter you will meet each of these controls and learn how to use each one to accomplish real work.

Each control item represents a certain kind of functionality. In some cases, the functionality is unique to the Find window—that is, you must use this window to control that particular aspect of a search operation.

In other cases, such as with the four buttons at the top right of the Find window, a direct correspondence exists between the button in the Find window and commands in the Search menu. That means you can get significant work done without opening this window. I'll point out the precise relationships between the Find window controls and the Search menu commands as I discuss how you use these tools.

Let's start with how to find an item quickly and easily without even looking at the Find window.

## Finding Selected Text

Many times you see a word in your code, such as a variable name, and you want to find another spot where that same variable is used. The editor lets you find such selected text very easily.

First, select the text you want to locate. You can do this using any method supported in CodeWarrior, such as double-clicking the word, or using the Shift key in conjunction with the arrow keys.

Once you have done that, choose the Find Selection command in the Search menu (Ctrl+H) and the selected text becomes your target text automatically. The editor jumps to the next occurrence of the desired text that occurs in the file, and selects that text. The search begins at the insertion point and goes toward the end of the file.

A brief jaunt back into terminology: Remember that the target text is what you're looking for. When a Find operation results in a successful match, the editor jumps to that point in the file, and selects the found text. I'll call this a *hit*. A hit occurs when you find what you are looking for.

Now, back to the tools. The same text may occur in a file many times. If the hit you locate isn't the one you want, and you want to look for the next hit, you can issue a Find Selection command again (via the menu, toolbar, or keyboard equivalent). Or you can use the Find Next command in the Search menu (Ctrl+G). As a mnemonic for remembering this particular shortcut, you can think of the "G" key as part of "find aGain."

In this circumstance, the effect of Find Selection and Find Next is identical. Each gets you to the next instance of the target text. However, note a subtle distinction between these two commands.

The Find Selection command requires that you have some text selected in the active editor window. It makes that selected text the new target text, and then finds it. The Find Next command simply looks for whatever is *already* in the target text, so it requires that you have already set target text somehow. Find Next ignores any selected text you may have in the active window. In other words, Find Selection resets the target text. Find Next does not reset the target text; it uses the current target text. Table 5.2 summarizes the differences.

TABLE 5.2 The differences between Find Selection and Find Next

| Command | Requires | Target Text | Result |
|---|---|---|---|
| Find Selection | selected text | changes to selection | finds selection |
| Find Next | target text filled | remains unchanged | finds target text |

So, if you have no selected text, Find Selection won't work (because you have no selection). However, Find Next would work (if you had already filled the target text).

**TIP**

You can also search backward (that is, toward the beginning of the file from the insertion point). Use the Find Previous Selection (Shift+ Ctrl+H) and Find Previous (Shift+Ctrl+G) commands.

The Find Selection command sets the target text and initiates a search. You might want to set the target text without starting a search. For example, this is what you want to do if the source for the target text is in one window, but you want to search in another window.

To do this, choose the Enter 'Find' String command in the Search menu (Ctrl+E). This command sets the target text without starting a search, a very useful command for loading up target text for future use. CodeWarrior remembers the target text until you reset it, so you can load the target text well in advance of the actual search if you want.

You can use these commands to search in any editor window. You can load the target text in one window and search in another. Just activate the window you're interested in after loading the target text.

That pretty much takes care of simple searching. Now let's take the discussion a little deeper to see how to control a search more closely, and how to replace a hit.

## Finding in a Single File

CodeWarrior's Find features for searching in a single file are quite powerful. Although this section discusses search and replace in the context of a single file, these techniques apply equally in multiple-file searches. I'll remind you of this when I discuss searching in multiple files a little later. For now, continuing our journey from the simple to the complex, let's take a look at how to do a complete and focused search and replace operation inside a single file.

### Entering Search and Replace Text

To control the settings that affect exactly how a search proceeds, you must open the Find window with the Find command in the Search

menu. Some aspects of a search, such as range and scope, require that you set certain controls in the Find window.

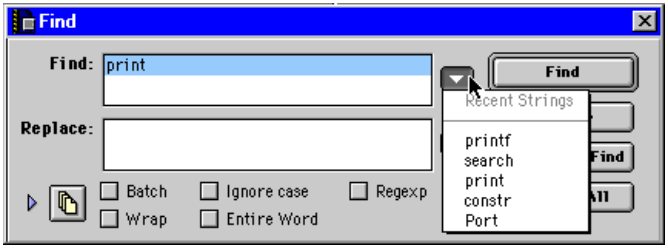
When the Find window appears, the target text is listed in the Find display. If you have already set target text with either the Find Selection or Enter 'Find' String commands, the target text appears here.

You can also enter the text directly in the display. Simply click in the target text display area, and type the string you want to find. The Find window supports copy and paste as well, so you can paste target text into the Find window.

CodeWarrior has one more nice feature to help you enter the target text. The editor remembers the five most recent target text strings. You can see what these are in the pop-up menu that appears to the right of the Find display area. Click the arrow button to display the menu. If you choose one of these strings, it appears as your target text. Figure 5.3 shows you this menu.

FIGURE 5.3

You can see up to five recent target text strings using a pop-up menu in the Find window.




The Find window displays the replacement text in the Replace area, just below the Find area. You can enter replacement text using all of the same techniques you would use for the target text. If you want to enter selected text from an editor window into the Replace field of the Find window, use the Enter 'Replace' String command in the Search menu (Shift+Ctrl+E). You can enter the replacement text manually in the Find window, or use copy and paste. The replacement text also has a pop-up menu that lists the five most recent replacement strings.

**TIP**

If you want to use the toolbar to enter a replace string, hold the Shift key down and use the Enter Find String icon.

## Searching with Special Characters

One way to enter the target or replacement text is to type the text directly into the Find window. You can search for anything you can type into the target text field, and replace it with anything you can type into the replacement text field. However, typing presents a bit of a problem.

The Find window intercepts Tab and Enter keystrokes. It uses the Tab key to switch focus from one item in the window to another. The window uses the Enter key as equivalent to clicking the Find button.

Nevertheless, you can indeed search for these characters, or replace text using these characters. CodeWarrior provides at least three ways around this hurdle, as follows:

1. You can select the target or replacement text in a source file, and then use the Enter 'Find' String or Enter 'Replace' String commands. Even if a Tab or Enter character occurs in the text, these commands put it in the Find window correctly.

2. You can copy the text from a source window, and paste it into the appropriate section of the Find window.

3. You can turn on the Regexp option and use the Unix-style convention of \t for Tab and \r for Return. However, if you turn on the Regexp option you should understand that it has consequences for how the search engine interprets the rest of the target text. Read "Using Regular Expressions" in the Cool Features section of this chapter for more information on the effects of choosing Regexp.

## Controlling Search Range

The next task you face is to determine the range of the search within the file. That is, what part of the file are you going to search in?

A search in a single file always proceeds from the insertion point. For this discussion, let's assume you are searching in a forward direction, that is, toward the end of the file. If the insertion point is not at the beginning of a file, you must decide if you want to search the whole file, or just the part of the file that comes after the insertion point.

Typically (but not always) you'll want to search an entire file. To do this, use the Find window's file wrapping feature. Click the Wrap check box in the Find window to toggle this feature on or off. If this option is on, you can start the search anywhere in the file. When the search reaches the end of the file, it wraps around to the beginning of the file and continues back to the original insertion point. If you only want to search from the insertion point to the end of the file, turn the wrap feature off. It's that simple.

**NOTE**

Although this discussion assumes a forward search, file wrap works whether you are searching forward or backward. If file wrapping is on, you can search the entire file regardless of the direction of the search.

Of course, another way you can ensure that you search a complete file is to place the insertion point at the beginning of the file before searching. You can scroll the window and click there, or type Ctrl+up arrow. However, moving the insertion point manually requires you to perform an unrelated action just to make sure your search covers the required range. That's not the easiest way to get the job done. The file wrapping feature is easier to use because you can set it once and forget about it.

## Controlling Search Scope

You have two other ways you can narrow or broaden the focus of the search. You can tell the editor to take case into account, which means that the editor considers whether or not a letter is uppercase or lowercase when determining if it matches the target text. You can also tell the editor whether to search for the target text inside other words.

You control the case sensitivity of a search with the Ignore Case check box. By default, the search process is case-sensitive—that is, the editor considers case. You can toggle this feature off and on by clicking the Ignore Case check box. When this option is on, case is ignored. This is a trifle odd, because you are turning an option *on* to turn case-checking *off*. A practical example should help here.

Assume you have a file that contains the text "codewarrior" (in all lowercase letters) somewhere inside it. If your target text is "CodeWarrior" and the Ignore Case check box is *off*, the search will not result in a hit because the editor is not ignoring case. "CodeWarrior" does not equal "codewarrior" when the search is case-sensitive. If the Ignore Case check box is *on*, then case is ignored. "CodeWarrior" does equal "codewarrior" in a search that is not case-sensitive, so the search results in a hit. The editor finds the word "codewarrior" in your file.

You control the word-sensitivity of a search with the Entire Word check box. If this option is on, the target text is treated as if it were a complete word. A successful hit must also be a complete word. Once again, an example will help to clarify.

As before, assume you have a file that contains the text "codewarrior" somewhere inside it. This time, your target text is "code" and the Entire Word option is on. A search does not result in a hit, because the entire word "code" does not equal the word "codewarrior" in this situation. If the Entire Word option is off, then a hit does result. Why? Because the string "code" appears inside the text "codewarrior" and so the search utility finds a match.

In other words, when Entire Word is off, the search algorithm searches inside words for the target text. When the Entire Word option is on, the search does not look inside words. By the way, a "word" is any collection of contiguous letters and numbers. A space or any form of punctuation—such as a comma, period, or asterisk—marks the boundary of a word. Just double-click some text and the editor selects what it thinks is a word.

Using the Ignore Case and Entire Word options judiciously, you can control the scope of your search quite precisely. You can broaden your search to find similar words that contain the target text, or narrow the scope to the precise target text.

There are a couple of other features of the Find window that I haven't discussed yet, such as the Batch and Regexp check boxes. Rest assured I'll cover them before this chapter is through. For now, it's time to get to the meat of the matter—actually finding and replacing text.

## Finding and Replacing Text

When it comes time to actually find and replace text, CodeWarrior provides you with six variations on a theme—commands you can issue to accomplish your goal. You can issue these commands using a variety of techniques. Table 5.3 summarizes all the options, and the different ways you can implement them.

TABLE 5.3 Find and replace commands

| Search Menu | Shortcut | Find Window Button |
|---|---|---|
| Find Next | Ctrl+G | Click Find (or press Enter) |
| Find Previous | Shift+Ctrl+G | Shift+Click Find (or press Shift+Enter) |
| Replace | Ctrl+= | Click Replace |
| Replace & Find Next | Ctrl+L | Click Replace & Find |
| Replace & Find Previous | Shift+Ctrl+L | Shift+Click Replace & Find |
| Replace All | n/a | Replace All |

As you can see, each command has an entry in the Search menu, and a corresponding button in the Find window. Let's look at what each one does.

### Find or Find Previous

After setting target text, and determining the range and scope of the search as discussed previously, you can find the target text. If the Find window is active, you can click the Find button, or press the Enter key. You can also use any of the other techniques listed in Table 5.3 to initiate a Find. CodeWarrior looks for your target text in the most-recently-active editor window.

If CodeWarrior doesn't get a hit, it beeps. If it does register a hit, the instance of the target text in the editor window is selected, and that window becomes the active window. This shifts the Find window to the background.

If you want to Find again from the Find window, you have to make it the active window once again. This is a clumsy approach, and CodeWarrior has a better way.

After setting up the Find window, you can switch to an editor window and run the Find from there as well. You don't have to return to the Find window. You can use the Search menu commands or key shortcuts to run the search and replace operation directly from the editor.

Remember, you can search forward or backward through the file.

**NOTE**

The Go Back and Go Forward commands in the Search menu are not related to search and replace operations. I'll discuss these commands when I talk about the browser in the next chapter.

### Replace

If you want to replace the text in a hit, you must have replacement text set up. As you know, you can do this directly in the Find window, or by selecting text and using the Enter 'Replace' String command in the Search menu.

If you have replacement text, and you have issued a successful Find command that results in a hit, simply issue a Replace command. You can switch back to the Find window to do this, but it is easier to simply work directly within the editor window.

When you replace text like this, the insertion point doesn't move, and you don't initiate another search. You just stay where you are in the editor window.

### Replace & Find Next or Previous

When you have a hit, and you want to replace it and find the next hit, use the Replace & Find command. Once again, you can switch back to the Find window to do this, but it is easier to simply work directly within the editor window. You have the option of working backward as well as forward.

**WARNING**

If you want to search a complete file, even if file wrap is on, you must continue to look for hits until you search the whole file. File wrap does not ensure that you will look at the entire file. It just means that the search process wraps to the beginning or end of the file.

### Replace All

After setting up the target text and replacement text, you can automatically find and replace every instance of the target text in a file. To do this, use the Replace All command. This is a powerful feature, and a great time saver.

Don't forget to consider the file wrap property of the search. In this case, because the search and replace is automatic, having file wrap on ensures that the entire file will be searched. If file wrap is off, only the part of the file from the insertion point to the end of the file is searched. When replacing all hits automatically, the search is forward only. You cannot search backward and replace all. If file wrap is off, this can make a difference.

**TIP**

When you are replacing all hits automatically, CodeWarrior opens each window in which a hit occurs. As it processes the hit, it replaces the text and updates (redraws) the window. You can make the process go much faster if you hide windows by dragging them partially offscreen or minimizing them before beginning the Replace All operation.

Use the Replace All command carefully. Make sure your search parameters, the target text, and the replacement text fully and precisely specify what you are looking for. If the settings are at all ambiguous, you could end up replacing more than you bargained for.

**WARNING**

The Replace All command is not undoable. If you discover that a Replace All operation did more than you intended, your only choices are to either manually fix the mistakes, or to revert your file to its last saved condition.

## Working without the Find Window

The Find window is not just a dialog box. It is a user interface element that you use to tell CodeWarrior how to run a search. Once you have set up the parameters, you can close this window with no ill effects.

You can begin a search in the editor at any time after you have set up your search parameters. As you know, you can even modify such

settings as the target text and replacement text directly from the editor, without returning to the Find window at all.

The real advantage of this design is speed and efficiency. You set up the parameters in advance, and then close the Find window. Otherwise the Find window just occupies screen real estate, and can hide the very things you are looking for. In addition, it is faster and more direct to work with Search menu commands or their key equivalents directly in the editor. Get familiar with the key equivalents of Search menu commands, and go for it. Before long you'll be a real wizard at finding and replacing.

You'll discover one more advantage to dismissing the Find window before beginning a search. Assume you have multiple editor windows open. If you start a Find from the Find window, it isn't always clear which window you are searching in until after the first hit. If you dismiss the Find window, you can be sure which editor window is the active window.

You've come a long way, but still more features remain to be discovered. Paramount among them is how to search in more than one file. Searching for hits in multiple files is a common task. You could manually switch windows and repeat a single-file search over and over again, but that's not terribly efficient. CodeWarrior has a better way.

## Finding in Multiple Files

The ability to search for target text in a whole series of files is vital to efficient programming.

Numerous reasons exist why you need the ability to search multiple files. Here's one common example. You know the name of a function in your project, but you can't remember what source file contains the code. How do you find it? You can look in all your source files, but you certainly want that search to be automatic. You don't want to open twenty or thirty or even a hundred files individually so you can look in each one.

CodeWarrior has just the features you want. Best of all, because it is built on the same search engine we have been talking about all along, you already know how to use most of the features. Everything you have learned about—entering target and replacement text, controlling the range and scope of a single-file search, changing directions—applies to a multi-file search.

There are a few additional considerations, however, and a few more controls to learn about. The range issue—how a file is searched and which files to search in—is a bit more complex, but even so, the whole process is pretty straightforward.

Fundamentally, all you have to do is specify which files you want to search in. CodeWarrior does the rest. Best of all, you can save whole sets of files for future reference. You'll learn how in this section.

## Activating Multi-File Search

The first thing you need to know about multi-file searching is how to turn it on. Click the multi-file search button in the Find window. A disclosure triangle appears to the left of the multi-file search button. Click this triangle to disclose or hide the controls related to multi-file searching. Figure 5.4 shows you the multi-file search button in its on state, and all the multi-file controls.

A scrolling display area lists the files in which you want to search. Above the display is a File Sets pop-up menu. This menu lists any file sets you have saved. I'll discuss how to save file sets in just a bit.

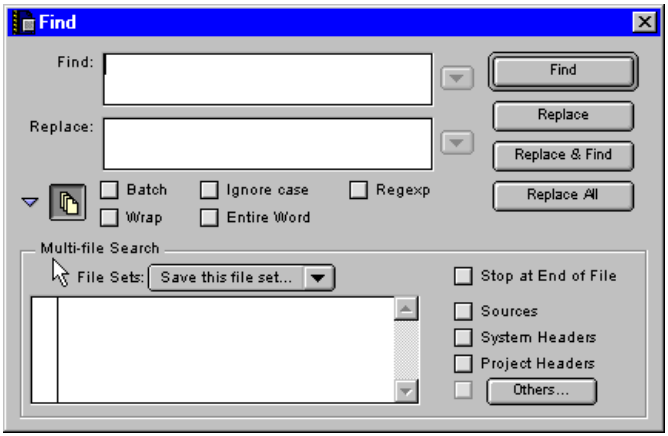
Several check boxes are available to the right of the file list. I'll talk about each of these boxes and their effects as I cover multi-file searches.

## Choosing Files to be Searched

If you want to search in multiple files, you must identify the files in which you want to look. You can add common groups of files automatically, or

FIGURE 5.4

Turning on multi-file search capability lets you search in a collection of files.




add any particular file at any time. You can also selectively remove files from the search list.

### Adding Common Groups of Files

The Find window controls give you some very direct and simple mechanisms for adding common groups of files.

Notice the three check boxes on the right side of the window:

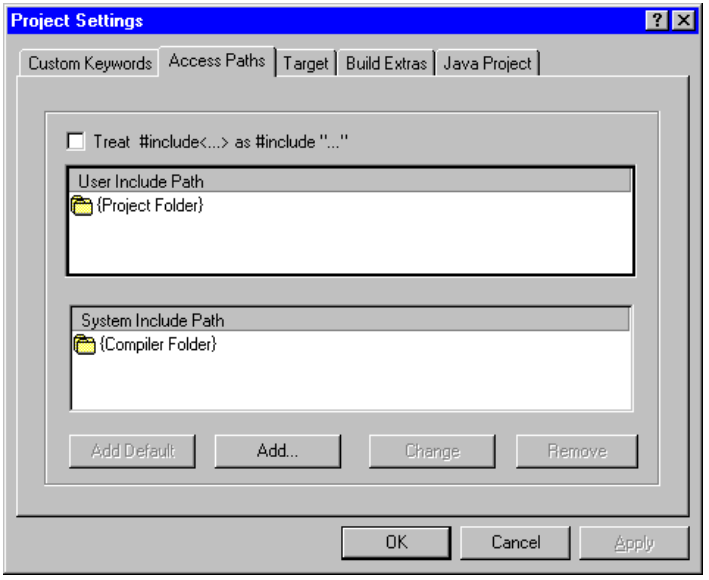
- Sources
- System Headers
- Project Headers

Each of these check boxes represents a group of files.

The Sources check box represents all the source files listed in the project. If you turn on this option, every source file in your project automatically appears in the search list.

The System Headers check box represents the header files included in your project that are found in System access paths. You control access paths in the Access Paths settings panel, shown in Figure 5.5. I discussed this panel back in the Setting Access Paths section of Chapter 3.

FIGURE 5.5
The Access Paths settings panel determines where CodeWarrior looks for files.




Typically, the system headers group includes general header files related to programming for the particular target you're interested in. You find these headers inside a folder inside the Metrowerks CodeWarrior folder. For example, the header files for the Microsoft Foundation Classes would typically be found in this group, if your project uses the MFC.

The Project Headers check box represents headers in your project folder. This group includes the header files you have declared for your own code.

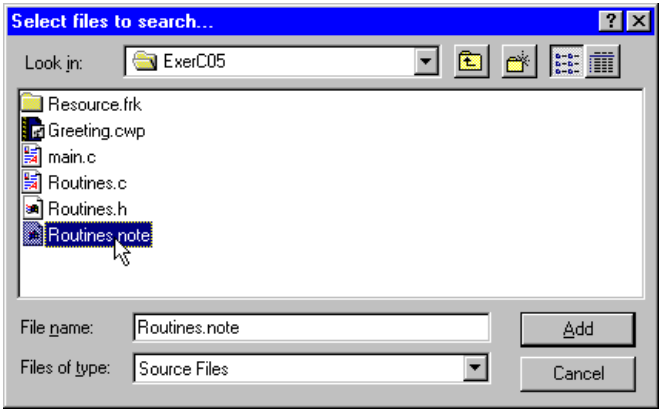
These three groups—sources, system headers, project headers—are the three most common groups of files in which you want to search. You can include these groups in any combination in the search list by checking or unchecking the corresponding control in the Find window. As you do, the files appear or disappear from the search file list. It couldn't be any easier.

### Adding Any Files

At times, you will want to search in files that aren't in any of the basic groups. The Others button at the bottom right of the Find window takes care of them. When you click this button, a dialog box appears in which you can browse your hard drive and add any file to the search list, as shown in Figure 5.6.

FIGURE 5.6

Use the adding search files dialog box to add any files to your search set.




You use the controls in this dialog box to navigate to a particular folder. Once inside the folder, you can add one or more files to the search file list. When your selection is complete, click the Add button to tell CodeWarrior that you are finished gathering files. CodeWarrior then adds the selected items to the search file list. To collect files from more than one folder, simply repeat the process.

### Removing Files

You can remove files from the search list in two ways. Click the check box to turn off any group—sources, system headers, project headers, or others. CodeWarrior removes all files in that group from the search list.

You can eliminate particular files as well, singly or several at once. In the search list, simply select the file or files that you want to remove. You can select any number of files using standard selection techniques. Click to select a single file, or use the arrow keys to move up and down the list. Shift+click to select a range of files, or Ctrl+click to select noncontiguous files.

When you have selected the file or files to delete, press the Backspace or Delete key. CodeWarrior removes the selected files from the list. It's that easy.

## Controlling Multi-File Search Range

As with a single-file search, you can start a multi-file search from either the Find window or from any editor window. The search proceeds according to the settings you established in the Find window.

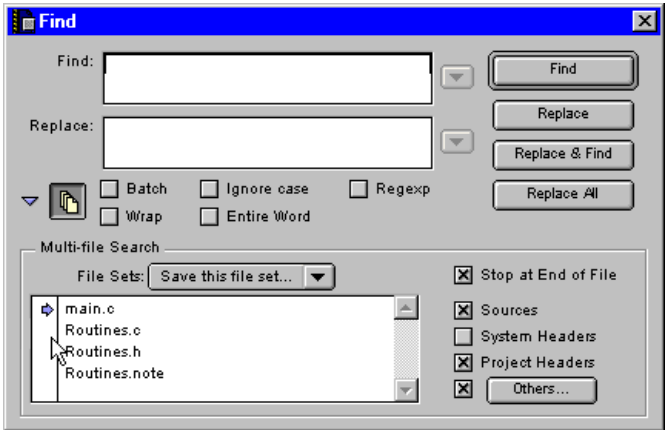
Pay close attention to the list of files in the search list. By default, the search begins at the start of the first file in the list, and continues to the last file in the list. This stays true regardless of what window is active when you start the search. The search does not start at the current insertion point in the active or most-recently-active editor window.

However, you can control which file is used as the starting point for the search. If you look at the list of search files in the Find window, you see a column to the left of the names. An arrow pointer in that column points to a file, as shown in Figure 5.7. This is the file about to be or in the process of being searched.

You can click in the column to the left of a file name to set this arrow marker. You can also drag the marker to a new location. When you

FIGURE 5.7

The arrow marks the current file in a search, and can be moved to control where the search starts.




reset the marker, you are telling CodeWarrior to start the search (or resume the search) with the marked file. All other files higher up in the list will be searched only if you have the file wrap setting on.

When we discussed single-file searching, you learned that the Wrap check box in the Find window determined whether a search went back to the beginning of the *file*. When you are doing a multi-file search, the Wrap check box controls whether the search returns to the start of the *file list*. If you want to ensure that all files are searched all the time, make sure you turn on the Wrap check box.

## Controlling Multi-File Searches

All the standard search commands work exactly the same for a multi-file search as for a single-file search. You can find, replace, replace and find, or replace all. You can search forward or backward.

**WARNING**

If you are using the Replace All command, CodeWarrior automatically searches through all the files, and replaces every hit. Because this is undoable, you want to be *very sure* you have your search parameters set correctly.

One more control that affects how a search proceeds is the Stop at End of File check box. If this option is on, then CodeWarrior (predictably) stops at the end of each file in the list when it is done searching that file. To continue searching in the next file, you must choose the Find in Next File command in the Search menu (Ctrl+T). If you are doing a manual multi-file search, you can use this command at any time to skip the rest of the current file and go to the next file. If you are replacing all hits, CodeWarrior stops at the end of the file. You can then continue the automatic search with the Find in Next File command, or stop.

## Working With File Sets

Once you begin working with the multi-file search features, you soon realize that you can spend a fair amount of time building up exactly the right list of search files for a particular project. This list may include a variety of files not attached directly to the project.

In addition, you may have a common set of files you use regularly. You should be able to set the search list to this common set of files easily, without having to go through the manual process of clicking the Others button and adding the files one at a time.

You can do these things by saving a search file set. A file set is a kind of shortcut—an easy way to set the search file list to a set of files you use regularly. You can create sets specific to a project, or make sets of files globally available to any project.

### Creating a Set

After you have a search file set exactly as you like it to be, look in the File Sets pop-up menu in the Find window. Figure 5.8 illustrates this menu.

To create a set, choose the Save this File Set item from the top of the menu to display the dialog box shown in Figure 5.9. You name the set, and then tell CodeWarrior whether you want to save this file set globally or for this particular project.

FIGURE 5.8

The File Sets pop-up menu shows you all available file sets.

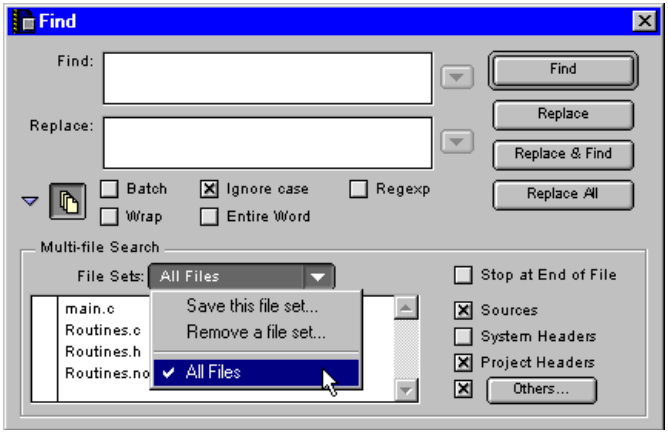
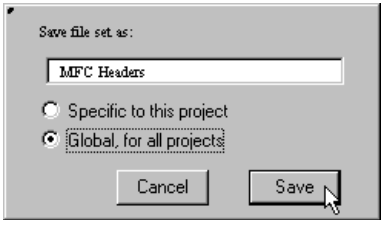



FIGURE 5.9

The Save File Set As dialog box lets you preserve a collection of files so you can reset the search list easily.




Simply click the appropriate radio button to save the file set the way you want. If you expect to use this set in many projects, click the global button. The name of this set then appears in the File Sets pop-up menu for all projects. If you save the file set as specific to the particular project, the set appears in the File Sets pop-up menu for the current project only.

**NOTE**

A good example of a global file set might be the header files for the Windows SDK (Software Development Kit). By setting these files up as a global file set, you can quickly search for declarations and prototypes in these headers no matter what project you're working in.

### Choosing a File Set

When you want to search in a particular set of files, just choose the name of that set from the File Sets pop-up menu. All the files in that saved set appear in the search file list. Any files that had been in the search set are removed. In other words, choosing a file set completely clears the file search list, and then resets the list to match the file set.

After you have selected a file set, you can then add the sources, system headers, project headers, or additional other files back into the set if you wish. However, the whole purpose of a file set is to save you from this kind of work. You can save these files in the set to begin with.

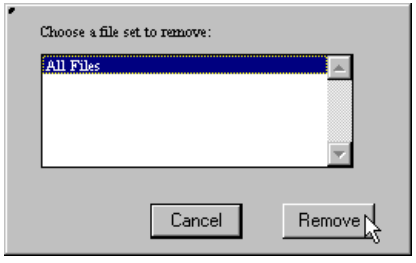
So, set up your sets to be complete and comprehensive. Then, when you choose a file set, all the files you want will be ready for searching.

### Removing a File Set

You can easily delete a file set from the File Sets pop-up menu. Choose the Remove a File Set item from the File Sets pop-up menu. When you do so, a dialog appears asking you to specify which set to delete from those available. Figure 5.10 shows this dialog box. Select the file set you want deleted from the list. You can only delete one set at a time.

FIGURE 5.10

You remove a file set by selecting it from the list and deleting it.




Deleting a file set from the list does not remove the associated files from the search list. It removes the file set from the File Sets pop-up

menu. After you remove a file set, you will no longer be able to choose that file set as a shortcut for adding a group of files to the search list.

# Cool Features

What you have seen so far are the kinds of features you would expect in any decent search engine—the ability to search and replace text at will in single or multiple files. The file set feature is a particularly nice twist on multi-file search management, and is something you are sure to use regularly. In other circumstances or in another editor, it might even qualify for the elusive title of "cool feature."

However, the CodeWarrior search engine has two more very powerful mechanisms you can use to really increase your productivity—batch reports and regular expressions. These features make file sets pale in comparison. Both batch reports and regular expressions qualify as truly cool features. Let's talk about each of these in turn.

## Using Batch Reports

Until now, I have talked about how to do manual searches, or an automatic, all-or-nothing replacement operation. Manual searches can be tedious, particularly if a file contains multiple instances of the target text. In many cases, you only want to work on a few of these hits, not every one. Still, you have to march through the files, one at a time, hit-by-hit, looking for the few items you really want to work on. In this kind of situation, a replace-all operation is inappropriate, to put it mildly. In fact, using the Replace All command when you only want to operate on a few instances of target text is a direct path to disaster!

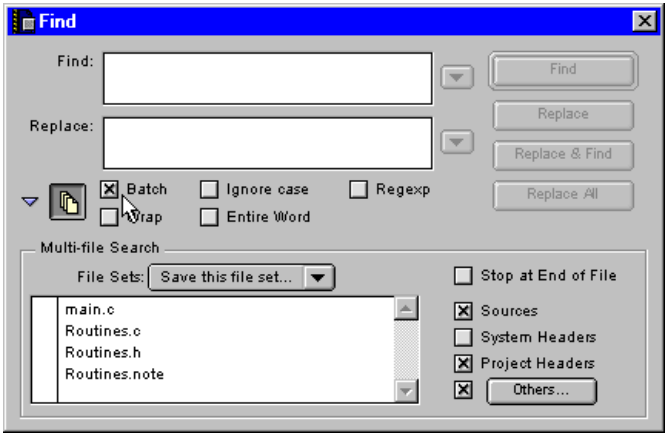
CodeWarrior has a powerful and effective middle ground between the tedium of the manual search and the no-turning-back power of Replace All—the batch report.

If you tell the search engine you want a batch report, rather than opening every file and marching you through the hits one-by-one, CodeWarrior presents you with a comprehensive list identifying each and every hit. You can then see each hit in context, decide if it is of real interest to you, and go to that hit instantly. Let's see how this works, because you will want to use batch reports often.

### Activating Batch Reports

As always, the first thing you want to know is, "How do I turn it on?" For batch reports, the switch is the Batch check box in the Find window, as shown in Figure 5.11.

FIGURE 5.11
Turn on batch reports to see a list of every hit in a search.




When you turn on batch reports, CodeWarrior no longer displays each hit in sequence in the corresponding editor window. The Replace All button is also disabled. You cannot use Replace All when you are doing a batch search.

To get a batch report, just issue a Find command. The process works for single-file or multi-file searches. Everything you have learned about the search engine still applies—scope, range, available commands, the works. You can issue the command from the Find window, or from the Search menu. It doesn't matter. As always, whenever you

issue a Find command, CodeWarrior uses the information you have set in the Find window, whether the Find window is open or not.

As soon as it finds a hit, CodeWarrior displays a Search Results window that contains (ultimately) a list of every hit. As the search continues, the window contents update until the search is complete.

### Understanding the Search Results Window

The Search Results window is one incarnation of the CodeWarrior message window. The message window is one of the coolest features in all of CodeWarrior.

You can use the Search Results window and its siblings (whom you'll meet later) in a variety of circumstances to view lists of code-related information. This is the first time you have met a message window, but it won't be the last. Other incarnations of the message window are slightly different, but they all follow the same general interface, and fulfill the same purpose—to present you data in a useful and efficient way.

## The Thought behind the Message Window

Modern programmers are faced with a real problem—software development systems are file-based. You put your code into a collection of files, and build software from the code in those files.

But the individual files are not really independent and self-contained. On the contrary, they are highly interdependent. The sum total of all the information in all of your files is really a huge interconnected database of knowledge, like a web of synapses. The division of this knowledge base into a collection of files is really artificial. While we all try very hard to make the contents of the files logical, the boundaries between files are often fuzzy. You ask yourself, "Does this code belong in this file, or in this other closely related file?"

continued ➠

## The Thought behind the Message Window

These kinds of arbitrary separations lead to problems. When it comes time to look for a bit of information, you can't remember where it is. No matter how logical your organizational structure, some pieces just don't fit in your designated niches. Further, in projects involving several programmers, everyone has a different idea about how things should be organized. You end up wasting precious time hunting for things that should, by rights, be at your fingertips.

The bottom line is, navigating through a large code base becomes a nightmare of confusion. You are lost in a sea of files, with no compass to guide you. The message window is designed to solve this problem.

Your code base is really a continuum of information. A tidbit of data may appear in several files. Why should you have to open up every single one of those files, and work with each file individually? The message window is a tool that says to you, "Based on a particular filter (in this case your target text), I have found every instance of what you are looking for. Here it is for your perusal and enjoyment."

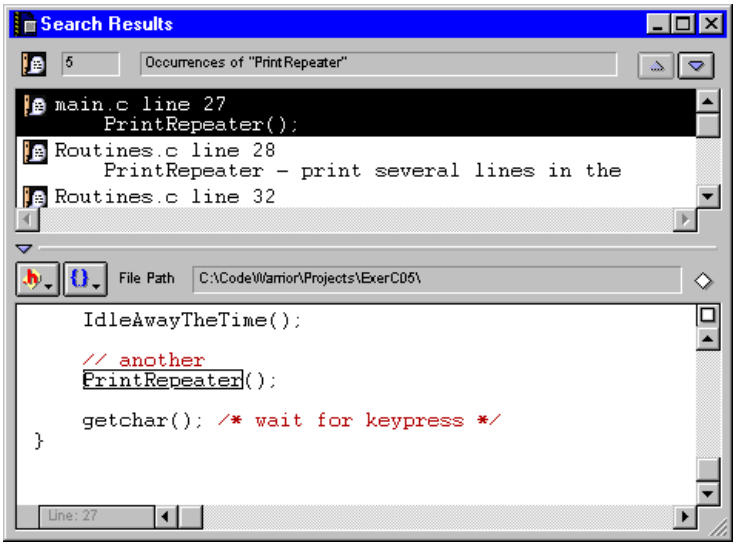
The message window gives you the list. When you select one of the items in the list for further examination, it presents that data to you directly. You do *not* have to open the source file. CodeWarrior goes and gets the data for you. You can edit or modify that data as you wish. CodeWarrior then saves the changes for you in the original source file. It's all automatic, completely transparent, and absolutely marvelous.

Artificial file boundaries are no longer barriers. The navigational nightmare becomes a sweet dream of an afternoon sail into a calm harbor. You'll see this same marvel again in CodeWarrior—in the code browser, and for compiler and linker errors. This approach to code navigation is the future of programming tools, and CodeWarrior takes you there now.

Figure 5.12 shows you the Search Results window that appears after a successful batch search.

FIGURE 5.12

The Search Results window displays the results of a batch search




At the top of the Search Results window, two display panels appear. One tells you the number of hits, while the other reminds you what you were looking for. To the right of that, at the top-right corner of the window, are two buttons—the Next and Previous buttons. You can use these buttons to walk through the list forward and backward, one item at a time. You can also scroll through the list using the cursor control keys or the mouse to select the hit that interests you.

Below these simple controls are two panes of information. In the top half of the window is a display that lists every hit. Each item in this list displays a noneditable line of code that contains the hit. You cannot change the contents of the list in any way, except to select one or another hit, but that's the fun part.

When you select an item from the hit list, the code for that hit appears in the source code pane at the bottom of the Search Results

window. This is not a static display; you are looking at the actual source file. The code in this pane is fully editable!

In this pane you see some old friends from Chapter 4 on the editor, including the function and interface pop-up menus. These work exactly as they do in a regular editor window. In fact, all the controls around the source code pane are just like those in a regular editor window, and work exactly the same way. You have a line number button, the file path telling you which file you are working in, scroll bars, and so forth. You can even control the pop-up menus in the same way, using either large icons at the top of the pane, or small icons at the bottom so you can see a few more lines of code.

You can resize the split between the panes by dragging the resize bar. You can even hide the source code pane entirely with the disclosure triangle to the left of the resize bar.

The Search Results window is an entirely new interface element, yet the only new items in the window are the Next and Previous buttons—everything else is familiar territory. The first time you see this window, you know intuitively what to do.

### Editing in the Search Results Window

To see the source file that contains the hit, just select a hit in the list in the top part of the Search Results window. The associated file appears in the source code pane, and is automatically scrolled to the location of the hit.

At this point, feel free to edit away. You have already learned about editing in CodeWarrior, and all the same tips, tricks, and techniques apply here. You can also use the Search menu's Replace command in this pane to replace the target text.

If you make a change to a source file, you can save the file manually at any time. If you forget to save the file, and you select another hit that requires CodeWarrior to display a different source file, CodeWarrior asks you whether you want to save changes. You cannot inadvertently lose changes you make in this window.

**WARNING**

Because the Search Results window is a two-pane window, only one pane is active at a time. You can change focus from the hit list to the source pane using the Tab key. However, you can't switch back to the hit list that way. If the source pane sees the Tab key it thinks you are typing code. This can be dangerous, because you can inadvertently replace selected text in the source pane with a Tab. The safest bet is to use the mouse to move back and forth between panes.

It gets better. Let's say you have done a batch search, and you have your list of hits. But gee, wouldn't it be nice if you could compare those hits to a slightly different batch search? You can.

Go back to the Find window, set up another search, and go for it. You'll get a second Search Results window. You can have multiple search results visible simultaneously, which means that you can copy and paste code between source panes, compare results, and have a grand old time. You are limited only by available memory.

**TIP**

You can also save the hit list if you want. While the Search Results window is the active window, and the hit list is the active pane, choose the Save a Copy As item from the File menu.

Just think about the power you have at your fingertips here. You don't need to know the name of the source file, where it is, or what else is in the file. All you have to do is decide what you're looking for, find it, and select a hit. CodeWarrior does the rest. It presents you with everything you need to know to work with the target text no matter what file it's in. The Search Results window is a tidy little package, neat, sweet, and complete.

## Using Regular Expressions

An end to the search and replace features is just around the corner, honest. In fact, this is it, the last major feature of the CodeWarrior search engine—regular expressions.

What is a regular expression? Well, if you have heard of grep, this is it. You use regular expressions to find text that matches certain criteria. You specify the search criteria using a special syntax. For example, let's say you want to find every instance of the word "window" followed by a single digit, such as "window1" or "window8." But you don't want to find just plain "window."

You can't do this by entering a simple search string. You would have to search ten times, once for each possible digit. Ouch! Or, you could so a search for the word "window" and perhaps get all kinds of hits that don't interest you.

However, you can perform exactly this kind of search automatically using a regular expression. Regular expression syntax allows you to specify variable characters in certain positions in a string, and control the acceptable range of those characters. You can say you would be willing to accept digits, or a range of letters, or any character or number of characters at a particular position in the target text.

Regular expressions can become quite complex and powerful. They can also be quite obscure and difficult to decipher. I'll keep this discussion simple.

### Activating Regular Expressions

Like other features in the search engine, regular expressions are controlled by a check box in the Find window. If you want to use regular expressions, turn on the Regexp check box in the Find window, as shown in Figure 5.13.

FIGURE 5.13

Activate regular expression syntax if you want to use regular expressions to search for variable text.

When this option is active, CodeWarrior interprets the target text using regular expression syntax, in which case certain ordinary characters such as a period or asterisk take on different meanings. You should only use the Regexp option if you understand regular expression syntax and are actually using it in your target text.

**TIP**

When using regular expressions, make sure you pay attention to the Entire Word option to ensure a clean search. Turning it on can have a nonintuitive impact on the regular expression search (as if anything about grep is intuitive).

### Understanding Expression Syntax

As just mentioned, several characters take on special meaning when you use regular expressions. Table 5.4 lists each character and provides a brief summary of its purpose in regular expression syntax.

TABLE 5.4 Special characters in regular expressions

| Character | Purpose |
|---|---|
| period (.) | Accept any character |
| asterisk (*) | Accept zero or more occurrences of the expression to left |
| plus sign (+) | Accept one or more occurrences of the expression to left |
| hyphen (-) | Specify a range of characters |
| backslash (\) | Interpret a special character as a nonspecial character |
| caret (^) | Specify an exclusion list (accept anything but); also, accept expression only if it is at the beginning of a line |
| brackets ([ ]) | Specify a list for inclusion or exclusion |
| parentheses (()) | Treat contents as a single expression for purposes of * or + |
| dollar sign ($) | Accept only if expression occurs at the end of a line |
| ampersand (&) | In the replacement text, include target text at this location |

If one of these characters appears in the target text, the search engine interprets it according to regular expression rules. This discussion of regular expressions is not exhaustive. It is intended to give you a taste for the power and flexibility you can achieve using regular expressions.

The best way to understand how regular expressions work is by example. In the rest of this section I'll give you a series of short tables, one for each special character. Each table has examples of the use of the character, and strings that match and don't match. Let's start with the period.

**Period** In typical use, the period character means you are willing to accept anything at that spot in the target text except a newline character. Table 5.5 shows some examples.

TABLE 5.5 Period examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| .core | score, 9core, #core | core |
| c.t | cat, cot, cut | car |

Acceptable hits include any character at the location of the period, but all other characters must match. Note in the .core example that the text "core" is not a hit, because it does not have a character in the position of the period in the expression.

**Asterisk** The asterisk says you will accept zero or more repetitions of the expression that appears just to the left of the asterisk. Table 5.6 shows some examples.

TABLE 5.6 Asterisk examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| 4*score | score, 4score, 44score, fourscore | encore, scare |
| .*score | score, fourscore, scores | core |

In this case, the principal part of the target is the text "score." In the first example, the actual text "score" matches because you will accept zero repetitions of the "4" character. Clearly, "4score" and "44score" also qualify. The word "fourscore" also qualifies, because it has zero occurrences of the "4" character. In the second example, the expression combines the period with the asterisk. Essentially, what you're saying is you'll accept any word that contains the text "score" because the expression says you'll accept any number of any character.

**Plus Sign** The plus sign works just like the asterisk, except you must have at least one or more repetitions of the expression just to the left of the plus sign to get a match. See Table 5.7 for a few examples.

TABLE 5.7 Plus sign examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| 4+score | 4score, 44score | score, fourscore |
| .+score | fourscore, fivescore | score |

Notice that the word "score" matches **4*score** in the previous example about the asterisk, but does not match **4+score**. That's because the plus sign requires at least one instance of the desired character, and the asterisk does not.

**Parentheses** If an expression is enclosed in parentheses, the editor treats it as a single expression for purposes of the asterisk and plus sign. Table 5.8 shows some examples.

TABLE 5.8 Parentheses examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| (ab)*c | c, abc, abbc, ababc | |
| (ab)+c | abc, ababc | c, abbc |

The parentheses in this example means that you have to have the pair of characters "ab" in that order (essentially saying that you want to treat them as a single character). Effectively, the expression (ab)*c finds every instance of the letter "c" because the asterisk says you'll accept zero instances of text before the letter "c." The expression (ab)+c requires that the letters "ab" appear in that order right before the letter "c."

**Brackets and Hyphens** Use brackets to specify a list of characters acceptable at a place in the expression. You can use a hyphen to specify a range of ASCII characters. See Table 5.9 for some instances.

TABLE 5.9 Bracket and hyphen examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| win[569] | win5, win6, win9, win95 | win4 |
| ba[a-z]k | bark, bank | baRk, ba9k |
| window[0-9] | window1, window6 | window10 |

The win[569] expression results in a partial hit on the text "win95"—the hit will be "win9," which matches the regular expression. Note that the word "baRk" does not match the expression **ba[a-z]k** because, in ASCII code, the letter "R" (uppercase) is not within the range a-z (lowercase).

**Caret** Where brackets give you the power to create an inclusive list of *acceptable* characters, use the caret inside the brackets to specify a list of characters *unacceptable* at a place in the string. Once again, you can use a hyphen (minus sign) to specify a range of ASCII characters. Table 5.10 gives some examples.

Note that "baRk" matches the expression ba[^r]k because an uppercase "R" is not a lowercase "r."

The caret has an additional meaning. If you use the caret before an entire expression, what you're saying is that an expression must appear at the beginning of a line. See Table 5.11 for examples.

TABLE 5.10 Caret examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| win[^569] | win1, wind, , win7, | win5, win6, win9 |
| ba[^r]k | bank, bask, baRk | bark |

TABLE 5.11 More caret examples

| Expression | Matches (if starts line) | Doesn't Match |
|---|---|---|
| ^the cat | the cat runs | feed the cat |
| ^if\( | if(, if(x==y) | only if, if ( |

In the second example, note the use of the backslash character before the parenthesis. That's because the parenthesis is a special character for regular expressions, and if you want to use it to mean you're looking for an actual parenthesis, you use the backslash to tell CodeWarrior to *not* interpret the parenthesis using its special, regular-expression meaning. Again in the second example, note that the text "if (" does not match because a space occurs between the word "if" and the parenthesis. The text "only if" does not match because the word "if" does not start the line, and the text does not contain the required parenthesis.

**Dollar Sign** Use the dollar sign after an entire expression to specify that an expression must appear at the end of a line. See Table 5.12 for some examples.

TABLE 5.12 Dollar sign examples

| Expression | Matches (if ends line) | Doesn't Match |
|---|---|---|
| .+the cat$ | Watch the cat | the cat eats |
| return;$ | return; | return; // end of function |

You can combine the caret and dollar sign to specify a complete line. For example, the expression ^**This is a complete line$** will match only

a complete line with that text, because the expression specifies that the line must start and end with the specified text. See Table 5.12 for some examples.

**Backslash** Use the backslash when you want the regular expression interpreter to read a special character as nonspecial. Table 5.13 shows some examples.

TABLE 5.13 Backslash examples

| Expression | Matches | Doesn't Match |
|---|---|---|
| final\. | final. | final6, final\, finality |
| array\[ | array[ | array, arraign |

As mentioned before, the backslash essentially turns off the regular-expression interpretation of a special character, with a couple of exceptions. The backslash tells the search engine to interpret the following character as a special character in the case of \t and \r. In these two cases, \t tells the engine to search for a tab character, and \r tells the search engine to look for a return. This is the only way to type these two characters—tab and return—directly into the target text. (You can copy and paste them into the Find string at any time.)

**Ampersand** Finally, we come to the ampersand. The ampersand has no significance in target text. It simply represents the ampersand character. However, it is special in replacement text. You can use it in the replacement text to specify that you want the find string replicated in the replacement text. Here's an example.

Suppose you have the expression **window[0-9]+** as target text. You will hit on every occurrence of the word "window" followed by one or more digits. The search engine will find the text "window92" as a hit. Now, suppose you have **my_&** as a replacement expression. If you replace the hit, the new text is "my_window92." The original found text is placed in the position of the ampersand.

That's the syntax for regular expressions. A few obscure exceptions to the rules exist, but I won't go into them here. The CodeWarrior documentation covers them fully. With these simple special characters, you can create very complex search filters.

**TIP**

Unless you are very comfortable with regular expression syntax, it is wise to test your search strings before relying on them blindly. Sometimes your expression can have bugs.

### Working with Regular Expressions

Simply enter the expression into the target text. CodeWarrior does the rest. If the regular expression interpreter does not understand your expression, you get an alert dialog box that tells you the nature of the error.

Everything else works exactly the same. Regular expressions are just a powerful way to specify a whole range of characteristics within the target text. You can use regular expressions as complex filters to find precisely what you are looking for.

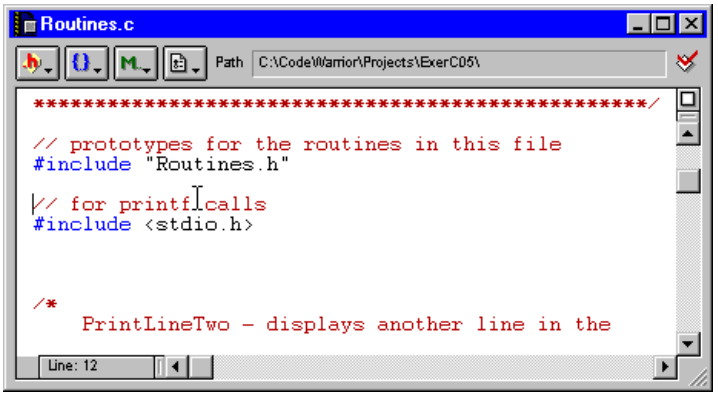
## Hands-On Exercises

The purpose of this exercise is to give you practice with the editor's search and replace features. To do this, you'll use prepared files that already include text that demonstrates these features. This experience will familiarize you with how to search and replace text in your own files.

**1. Launch CodeWarrior and open both a project and a file.** If CodeWarrior is not already running, launch CodeWarrior, and then open the Greeting.cwp project file in the exercise folder for this chapter, Exer05. After you have the project open, open the Routines.c file from that project.

**2. Find a selection.** In the Routines.c file, locate the first occurrence of the text "printf" near the start of the file. It's in a comment just above the line of code that reads **#include <stdio.h>**. Figure 5.14 shows the location in the file. Double-click the word "printf" to select it.

FIGURE 5.14
Locate the first occurrence of the phrase "printf" in the file Routines.c.




Once you have selected "printf," choose the Find Selection item from the Search menu (Ctrl+H). I strongly recommend using the accelerator because the keyboard is generally a much faster means of controlling a search than going to the Search menu repeatedly. Whichever technique you use, when you issue the Find Selection command the editor locates the next occurrence of the target text, which is inside the **PrintLineTwo()** function.

Now issue the Find Next command (Ctrl+G). You'll find a third instance of "printf" in the **PrintRepeater()** function. The Find Next command works because you loaded the find string with the "printf" target text when you issued the Find Selection command just now.

If you have followed the instructions, you are now at the last occurrence of the target text in the file. If you try to find it again, the computer beeps because it has reached the end of the file without a

hit. However, you can search backward with the Find Previous command (Shift+Ctrl+G). Issue that command now, and you'll go back to the **PrintLineTwo()** function. Do it again, and you'll return to the comment at the start of the file.

**3. Open the Find window.** In this step you take a good look at the Find window. To open this window, choose the Find command in the Search menu (Ctrl+F), and the window shown in Figure 5.15 appears. Notice that the "printf" target text is already entered in the Find String section of the window. That happened when you issued the Find Selection command in Step 2.

FIGURE 5.15
The Find window already has target text entered.

Take a look around and familiarize yourself with the controls. Notice the edit fields to enter Find and Replace strings, and the check boxes to control range and scope.

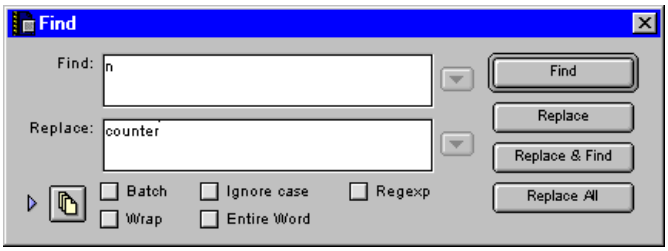
**4. Search for and replace text.** In this step you're going to find and replace a local variable named "n" with a more descriptive word, "counter." To demonstrate all the features of searching and replacing, please follow the instructions in this step carefully.

First, activate the Routines.c window and make sure the insertion point is at the beginning of the comment located just before the Idle-**AwayTheTime()** routine. The starting point is important so you can see the effect of file wrapping, among other things.

Now switch back to the Find window so you can set up the search. In the Find String section enter the letter **n** as your search text. Then, in the Replace String section, enter the word **counter**. Figure 5.16 shows you what the Find window should look like.

FIGURE 5.16

Prepare to search for the letter "n" and replace it with the word "counter."

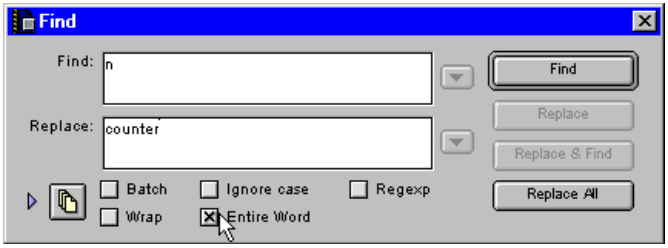



Now, click the Find button (or press the Enter key). Instantly the Routines.c window becomes active, and the letter "n" in the word "nothing" inside the comment is found. Unfortunately, this isn't what you're looking for. Clearly this is an inefficient way to search for a variable named "n" because you're going to find every single occurrence of the letter in every word in the file. Ouch!

The way around this is to control the scope of the search and limit it to a search for the *word* "n" rather than the *letter* "n." Go to the Find window, and turn on the Entire Word check box. Figure 5.17 shows you what the Find window should look like, with the Entire Word option activated. Then click the Find button again.

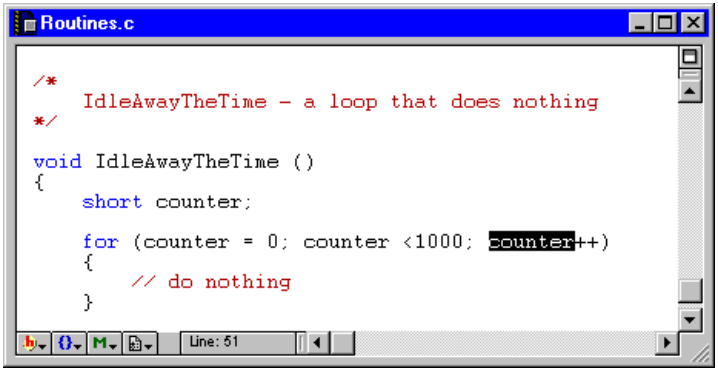
This time, the second letter "n" in the word "nothing" in the comment is skipped, and the local variable **n** is found inside the code. This is what you're looking for, so use the Replace and Find Next item (Ctrl+L) to replace this text and find the next instance of the target text. Continue to issue this command until you can find no more instances of the variable **n**. The computer will beep when you have an unsuccessful Find operation. All told, you should find three more occurrences of **n**, each inside the **for** statement, and you should replace each with

FIGURE 5.17
Limit the search so it finds whole words only.




the new name **counter**. Figure 5.18 shows you what the code looks like when you're all done.

FIGURE 5.18
The results of the initial phase of the search and replace mission.




**5. Search the entire file.** So far, so good, but remember that you started this search in the middle of the file. There could be other instances of "n" earlier in the file that you want to change. You can search backward, but an easier way to handle this is to turn on the file wrap feature for the search.

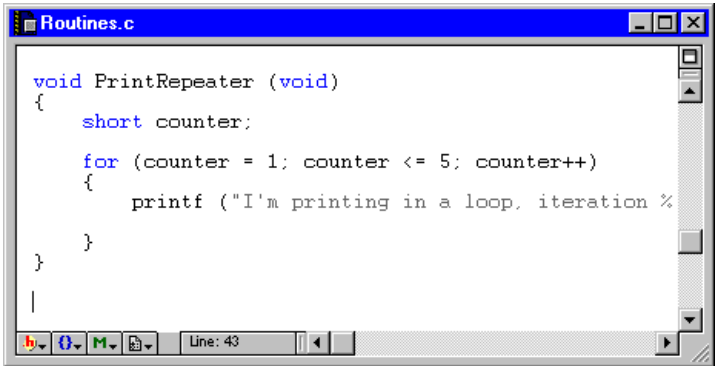
Go to the Find window, and click the Wrap check box to turn it on. Then begin another search. You don't have to move the insertion point. Now that file wrap is on, even though the insertion point is at the end of the file, the search will continue at the beginning automatically.

The first instance of "n" you find represents a newline character in the **PrintLineTwo()** function. You don't want to replace that. So find the next occurrence (Ctrl+G). You jump to the **PrintRepeater()** function, and once again find yourself with a loop counter variable named **n**. This is what you want to replace. Replace each instance of the variable **n** in this function, using the Replace and Find Next command (Ctrl+L).

Be careful! In some cases you do *not* want to replace the hit text with the target text. At the end of the **printf()** statement in this function, a **\n** newline character (don't replace that) is followed by the variable **n** (*do* replace that). Figure 5.19 shows the end result.

FIGURE 5.19

The **PrintRepeater()** function after completing the search and replace mission



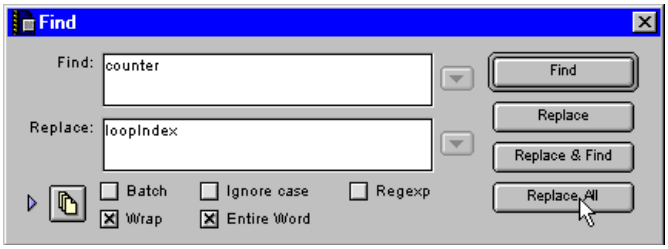

As you can see, if you had issued a Replace All command you would have received undesirable results, despite the fact that you narrowed the search as much as possible. However, sometimes replacing every instance of the target text is exactly what you want to do.

**6. Replace all instances of target text.** First, set up the search. In this case, let's replace every instance of the variable **counter** (all the changes you just made) with the name **loopIndex**, which is even more descriptive. Go to the Find window, enter **counter** as your Find String, and **loop-**

**Index** as your replace string. Figure 5.20 shows what the window should look like. Note that file wrap and word search are both still on.

FIGURE 5.20

Prepare to replace all instances of the word "counter."




Then click the Replace All button—done. The Routines.c window becomes active, and you should examine the contents of the window. Every instance of "counter" now reads "loopIndex." Because the Wrap feature was on, the entire file was searched. And because the Entire Word option was active, only the precise word "counter" was replaced. As you saw in step 5, these two limits do not *guarantee* that only those instances you want to find are hit, but they do go a long way toward focusing the search properly.

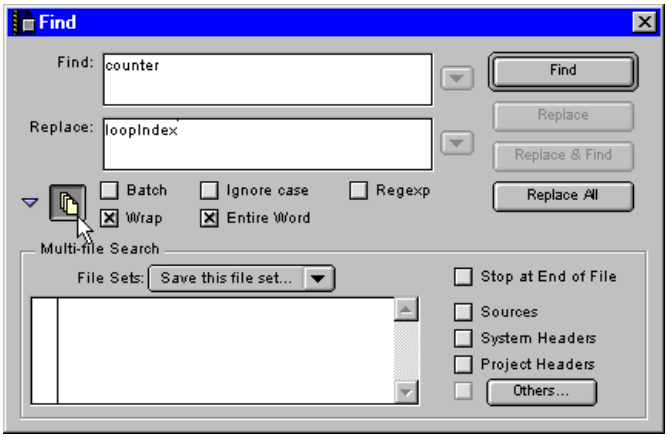
**7. Prepare to search in multiple files.** In this step you begin exploring the editor's multi-file search features. As a regular feature of multi-file searching you will want to search the project's include files. As you saw in the previous exercise, the project manager doesn't know about include files until after the compiler has had an opportunity to read the source files. So, compile all the code in this project with a Bring Up To Date command in the Project menu.

After that, the first task you have to accomplish is to turn on multi-file searching, so click the multi-file search button. Then click the disclosure triangle beside that button to see the multi-file search controls. Figure 5.21 shows you the complete Find window with all the multi-file search features visible. Notice that the multi-file search 3D button

is down. This is the button that activates multi-file searching, whether the controls are visible or not.

FIGURE 5.21

The multi-file search features on display in the Find window.




The next step is to decide which files to search in. In this case (as in many cases) you'll want to search in both your source files and your header files. Click the Sources and Project Headers check boxes to add those files to the search set, and the file names appear in the file list. In addition, let's add one more file that isn't either a source file or a header file. Click the Others button to display the standard file dialog you use to add files to the search list. Navigate to the Exer05 folder (if you're not already there) and double-click the file named Routines.note. This file has some documentation about each of the routines in this project. When you're all through, the Find window should look like Figure 5.22, with four file names appearing in the search file list.

**8. Save a file set.** You may want to search this same set of files on another occasion, and you just did some work setting up the file list, so it would be wise to save that work. Use the File Set pop-up menu at the top of the list, and choose the Save This File Set item as shown in Figure 5.23.

FIGURE 5.22

The list of files in which you intend to search appears in the bottom part of the Find window.

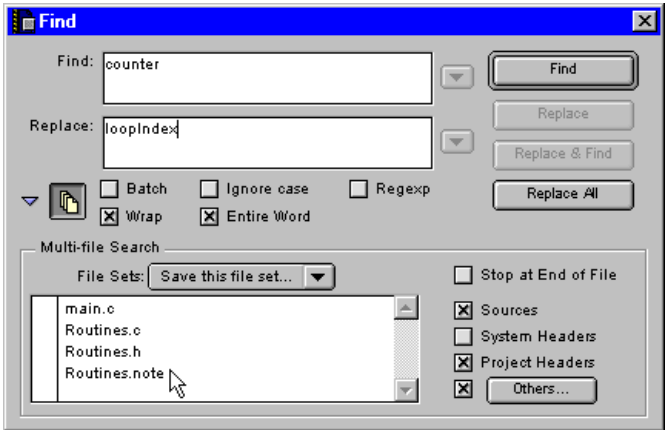
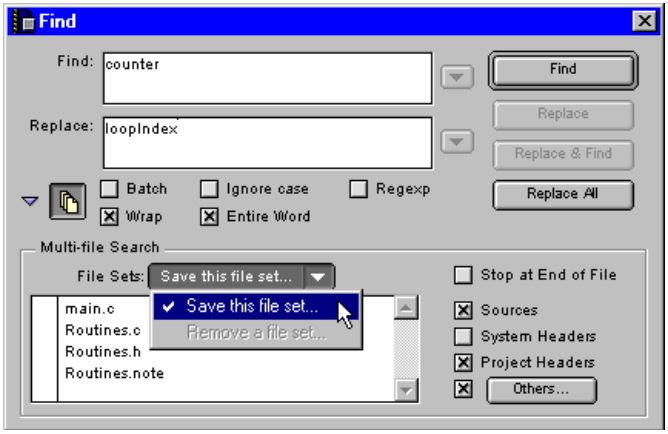



FIGURE 5.23

Save your file list for future reference.




In the resulting dialog box, give this file set a name, and save it as particular to this project. When you're done, that name appears in the pop-up as the currently selected set.

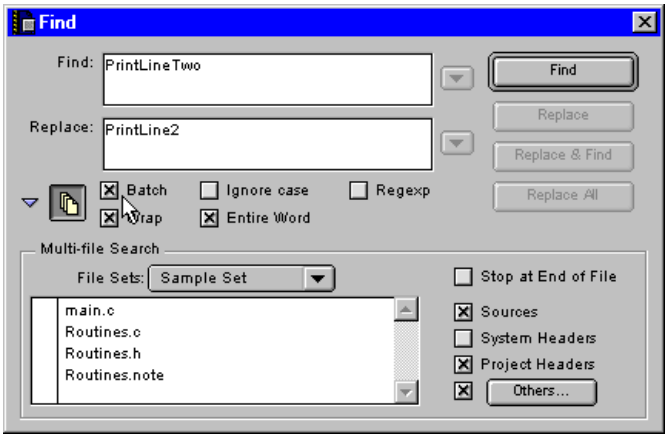
**9. Do a batch search for a function name.** In this step you look in every file in the list for references to a particular function. To set up this search, go to the Find window and enter the function name **PrintLineTwo** in the Find String section. In the Replace String section, enter

**PrintLine2** for the replacement text. You'll use the replacement text in the next step.

Finally, make sure you turn on the Batch option. Figure 5.24 shows you what the Find window should look like when it is set up correctly.

FIGURE 5.24

Prepare for a batch search for a function name.

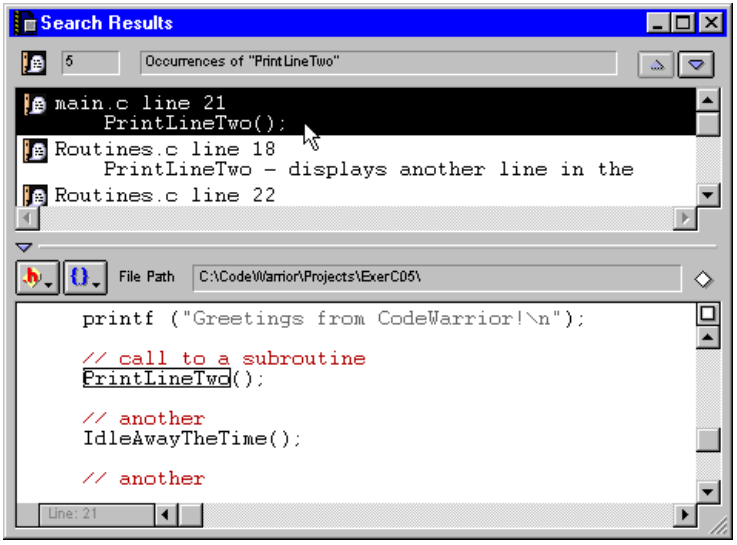



Once you're all set up, click the Find button, and every instance of the target text appears in the Search Results window, as shown in Figure 5.25. The window lists five hits. The first one is in the main.c file, and it is on display in the source pane.

At this point you should explore the Search Results message window to see how it works. Scroll through the list of hits, and watch what happens in the source pane. Use the interface and routine pop-up menus in that pane if you wish, because the entire source file is available to you.

Pay particular attention to the Search menu when the source pane is active and the target text is selected. Don't change any text just yet, but do notice which commands are available to you. You can replace text (Ctrl+=) on a hit-by-hit basis if you wish, and you can use the Find

FIGURE 5.25
The results window for this search shows five hits.



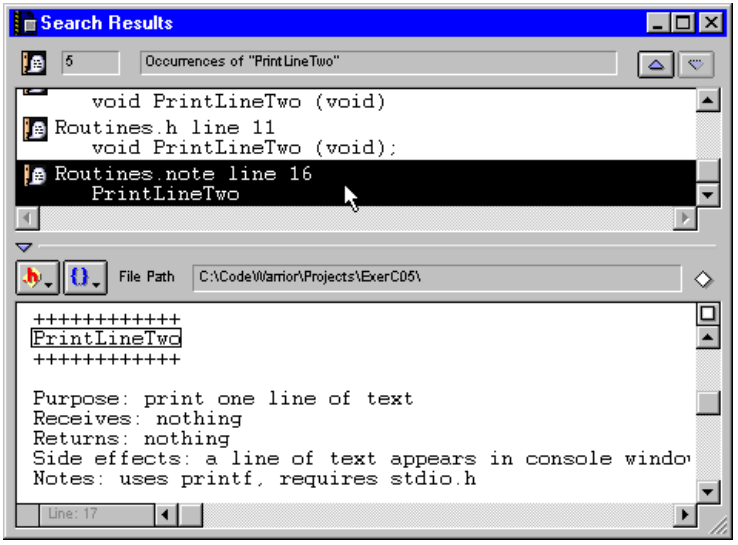

Selection command. But you cannot Replace and Find Next, so you do not have the easy facility to search for the next or previous occurrence of the target text in a file. Use the list at the top of the window to move between hits. This feature illustrates the fact that the best purpose of the batch search is to filter through a large number of possible hits to find the few that interest you.

For example, let's say you were interested in the notes you had created for this function. In the list of hits, you would scroll to the hit found in the Routines.note file and display that hit, as shown in Figure 5.26.

**10. Do a multi-file global replace.** For the ultimate in power and thrill-seeking danger, you can replace every instance of the target text in every file in your multi-file search list. Let's see how it's done.

First, go back to the Find window and turn off the Batch option. Notice that the Replace All button becomes active when you do. Everything else can be left the way it was for step 9—finding

FIGURE 5.26
Find what you're looking for in the Search Results window.




"PrintLineTwo," replacing with "PrintLine2," and looking in the same set of files. When everything is ready, click the Replace All button.

In rapid succession, each file in which a hit occurs opens, and CodeWarrior replaces the text. When you're all done you have a series of open windows. If you want to save the changes, you can choose the Save All item in the File menu. You have successfully renamed the function in every place where it occurs—header, function calls, function definition, and even in the notes about the function.

**WARNING**

Of course, truly Bad Things happen if your target text is vague and text that you don't want to change is found and replaced. Use multi-file Replace All with caution, because it cannot be undone.

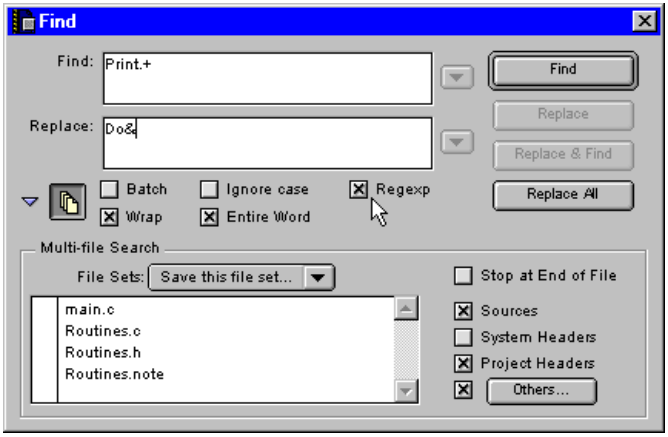
**11. Use regular expressions for fame and fortune.** This last step demonstrates the power of regular expressions. Follow along carefully, and

watch what happens. First, go to the Find window and set up the search parameters as follows:

- Find String: Print.+ (that's "Print," a period, and a plus sign)
- Replace String: Do&
- Options On: Wrap, Entire Word, and Regexp
- Options Off: Batch, Ignore Case
- Multi-file Search Button: On
- Search Files: main.c, Routines.c, Routines.h, Routines.note

Figure 5.27 shows you what the Find window looks like.

FIGURE 5.27
Prepare to use regular expressions.




If you want to be cautious, you can turn on the Batch option, click Find, and see the list of hits. If you'd like to be adventurous, tighten your seat belt and click the Replace All button. CodeWarrior goes into a frenzy of activity, and a brief moment later calm descends once again. If you look at the files, what you'll see is that every instance of

every function name beginning with the word "Print" (case-sensitive) has been changed so that it begins with the word "Do" (as in **DoPrintRepeater**).

The search string "Print.+" said to look for every occurrence of the word "Print" followed by one or more characters. The replace string "Do&" said to put the word "Do" on the front of each hit. Using case sensitivity ensured that calls to the standard library function **printf()** were not changed.

You may not need to use regular expressions often, but they are absolutely marvelous for those occasional searches where you want to find several different kinds of text that share some unique identifying characteristic. Mastery of regular expressions is a true sign of a CodeWarrior power user.

With that, you have completed another exercise and gained some real practical experience with the editor's search engine. At this time you can close all the files and exit CodeWarrior—until the next chapter.

## Summary

I bet you never thought that searching and replacing text could be such a big topic. When you think about it, though, you realize that a powerful search engine is a real boon to a programmer.

In this chapter you have seen how to use the CodeWarrior IDE's comprehensive search engine. You learned how to do a simple search by selecting text and pressing Ctrl+H. You mastered search and replace in a single file—how to look for words, parts of words, and special characters while moving either forward or backward through the text. You also saw how to control the extent of the search. Most important, you learned how to substitute new text for the text you found, either instance by instance or all at once. You even learned how

to do all this magic directly in the editor, without using the Find window controls.

You then extended these skills to working with multiple files. You can manage sets of files, and modify a file list quickly and easily. And you mastered how to preserve a file list for future reference, one of CodeWarrior's nicest features.

After all this, you still weren't through exploring the rich feature set! Next you turned to batch searches. You learned how to see all the instances of target text from anywhere in your project. You met the message window and know how to move through the list of hits. You saw how to replace and edit text directly in the message window quickly and easily.

Finally, the icing on the cake. For truly powerful batch work, you learned about CodeWarrior's support for regular expressions. You can use regular expressions to search for arbitrarily complex text strings without limiting yourself to specific characters.

Fantastic! And more is still to come. The next stop on the CodeWarrior journey is the browser. I'll save the real goodies for the next chapter. Let me just say that you are in for a real treat. The browser is the epitome of simplicity, power, and flexibility. You may not want to power down CodeWarrior after all.

# CHAPTER SIX

6

# Browsing Your Code

- Understanding the Browser
- Activating the Browser
- Accessing the Catalog
- Using the Graphical Browser
- Using the Class Browser
- Using the Symbol Window
- Navigating Code with the Browser
- Cool Features
- Setting Browser Options
- Hands-On Exercises

In the previous chapter you had a glimpse of the CodeWarrior message window and the programming concept it represents—that your code is an integrated database of information. You should be able to move freely between different parts of your code to see elements that interest you. You should not be restricted by arbitrary, file-based barriers.

The CodeWarrior browser takes this concept to new heights. In this chapter you'll learn how to use the browser to navigate through your code with ease. We'll take a high-level look at the browser design, so you understand the principles involved. Then we'll look at how you use the browser to get real work done. The browser's best features are designed for object-oriented code, but you can use the CodeWarrior browser for any kind of code.

## Understanding the Browser

The CodeWarrior browser is really two distinct elements—a parser, and a visual interface. A parser reads your code and responds or performs an action based on its understanding of that code. A compiler is a kind of parser in this generic sense because it reads source code and generates corresponding object code. The browser's parser reads code, identifies various elements in your code (such as functions, classes, and so forth), and determines the relationships and locations of each of these elements.

The parser works equally well for C, C++, Object Pascal, and Java. You never see the parser; rather, it works behind the scenes. The only thing you have to do is turn it on. You'll learn how in just a bit. After you turn it on, you don't have to run the parser separately. The parser is fully integrated into the IDE and works in conjunction with the CodeWarrior compilers.

## The Fundamentals of Object-Oriented Code

Object-oriented code is a topic much larger than a sidebar can cover. However, I can give you a brief overview of some basic object-oriented principles.

In traditional procedural code, you write a main program that calls subroutines that work on data. The data may be global to the program, or local to the routine. For the most part, any routine can call any other routine.

Object-oriented code is computer code that uses *objects*. An object is a (mostly) self-contained data structure. In most object-oriented languages, an object consists of both routines (often called methods or member functions) and variables. The object is designed so its variables contain the data that the object's functions need to get work done. As a result, data and functions are *encapsulated* in the same object.

Because objects are—for the most part—self-contained, software becomes modular. In a well-designed object-based system, you can plug in new or replacement objects and everything still works.

The next question is, where do objects come from? In most object-oriented languages, an object is an example or *instance* of a class. A class is a declaration of a structure that specifies the nature of the object. It declares both the data variables in the object, and the associated functions. Therefore, in simple terms, a class is a description of a group of functions and the data on which those functions operate.

An object is nothing more than an instance of a class in computer memory. A class, per se, never exists inside a program—only *instances* of a class exist. It's a bit like a cookie cutter. The class is the cookie cutter, and computer memory is the dough. You can stamp out an arbitrary number of objects using a single cookie cutter. Once an object exists in memory, you can modify the data inside the object at will.

continued ➠

## The Fundamentals of Object-Oriented Code

In a nutshell, a class describes an object and the object is an instance of the class. Functions belong to the class, not to the program as a whole. Likewise, the data on which these functions operate belong to the class. This encapsulation of functions with their related data is critical to object-oriented programming.

Object-oriented languages also implement a concept called *inheritance*, where one class inherits behavior from an ancestor class. Generally, a descendant gets everything the ancestor had, and then adds a little more or modifies the routines it inherited from the ancestor. This makes it possible to put a function that is common to a group of classes in an ancestor class. Then, any class that needs the function can inherit it from the ancestor.

The principle of inheritance is another fundamental underpinning of object-oriented code. It allows the programmer to write one function that is used in many places, and helps achieve the goal of code reusability. However, it makes finding a function very difficult. The function does not exist directly in every descendant, only in the ancestor. And that's one reason why browsers were invented.

The parser generates a catalog of information from your code. This catalog is a database of every symbol the compiler encounters as it reads and interprets your source code. The resulting catalog identifies the following eight kinds of data:

- Classes
- Constants
- Enumerated constants
- Functions
- Globals
- Macros

- Templates
- Typedefs

Information about all of these symbols is stored in the catalog, ready for your use. That's really all you need to know about the parser—it exists, and it reads your code to identify each different kind of data and how it fits into the whole picture.

The visual interface component of the browser is what this chapter is all about. This interface—what I'll call simply the browser from now on—is your pathway into the catalog—the database of code symbols. The browser gives you enormous power and flexibility in accessing your data.

Many features of the browser shine brightest with object-oriented code. That's not surprising. Finding just the right element in object-oriented code is more difficult than in procedural code. This problem comes from the very nature of object-oriented code, where information is distributed throughout a program. In fact, one of the fundamental principles of object-oriented design is called *data hiding*, which does not bode well for data finding.

The original class browsers were designed to help object-oriented programmers search through their classes and methods easily. In fact, many stand-alone browsers work only for a particular object-oriented language. You'll also note that such a tool is generically called a *class* browser, because a class is a fundamental unit in object-oriented programming.

The CodeWarrior browser is not so confining, and it is not strictly a class browser. It is a code browser. It understands all of the four supported languages, and you can use it to browse any sort of code, including procedural code written in C. When you are working with procedural code, some browser features won't be available, because they are designed strictly for object-oriented code. For example, a graphical representation showing class hierarchies requires classes.

## What Is Data Hiding?

Data hiding is an intriguing object-oriented concept. When you analyze how your software is going to work, you identify various kinds of data your code requires. The nature of the data depends on the subject matter of your code, but you are sure to use data somewhere.

Simply put, the entire program doesn't need to know about most of the data, because most information is particular to one or another small part of the code. One goal of object-oriented design is to identify which data goes with which parts of the code.

After you make the association, you put the data together with the routines that use the data into a single class. (The buzzword for this idea is *encapsulation*.) Then, using object-oriented programming syntax, you limit access to the data so that only those parts of the program that need the data know that it exists. In other words, you hide the data.

The real benefit you gain from data hiding shows up when you modify your program. Because significant parts of your program are completely hidden from other parts, you know that changes you make in one part of your code cannot affect other parts of the code. Why? Because they are hidden from each other.

Actually, the same concept works for routines as well as data. A really well-designed piece of object-oriented software can be visualized as a series of interconnected black boxes. Everything inside each black box (a module) is hidden from every other black box.

However, having both data and functions widely distributed like this can cause difficulties locating precisely what you're looking for when you navigate through your code, hence the need for a browser.

Nevertheless, as a database access engine, the CodeWarrior browser works for all kinds of projects, and all kinds of code. In this discussion

I'll use the terms *data* and *code* interchangeably, because the data in the catalog is, in fact, your code.

## The Thought behind the Browser

Your code is a mass of language-specific keywords combined with terms that you define—your symbols. A symbol can be a variable name, a function name, a named constant, a macro, a typedef, a class name, and so forth. In a sense, your code is really an ordered structure of symbols. You organize your symbols according to the syntax rules for a programming language, and they start to make sense.

Traditionally, you keep all this information in a series of files. You try to keep the file content organized logically, but your mass of data is separated into chunks based on the demands of a file system. It is extremely unlikely that the file system understands or cares about the content of your files. The file system isn't intended to give you free access to random contents scattered across a dozen files. It just keeps track of files, and lets you open one when you need it. It is up to you to know what's inside your files.

Time and time again you are forced to ask the question, "What file is this in?" Unfortunately, sometimes you don't know the answer. Then you are off on a hunt.

When you approach the code in your project, you aren't likely to have file-related questions as a paramount consideration. You are much more likely to want to know something like, "What's the value of that constant?" You aren't going to remember what file contains the definition for that constant. Similarly, you might want to know whether a function in a class is inherited, whether a variable is static, what the expansion of a code macro is, and so forth.

continued ➠

## The Thought behind the Browser

All of these questions have a common thread. They are all *code-related*, not file-related. After all, when you're working with code, aren't code-related issues going to be most important? Bottom line: When it comes to code you are far more interested in *what* something is rather than *where* it is.

Sorting through data is what computers are good at. Sorting through your code is what the CodeWarrior browser is designed to do. The whole idea behind the browser is to give you an easy way to answer code-related questions. The browser lets you find out what something is in your code. You no longer have to worry about where it is, or how to find it. You saw this happen with the message window. The barriers between files fade away. And *that* makes you more productive.

Like a good database access program, the browser does not dictate precisely how you must view your data. The browser gives you a variety of views into the database. You can mix and match these views as your goals require. Fundamentally, the browser has three different ways of viewing data:

- **Catalog**—for a comprehensive, broad view of all symbols
- **Hierarchy**—for an inheritance-based look at your classes
- **Class browser**—for a class-based view of your code

You can look at a very broad range of data (wide scope), or you can focus on a smaller chunk (narrow scope). The interface tools necessary to see a wide scope of data differ slightly from what you need when you focus on a small area of your code. So the browser has a different kind of window for each of the different kinds of views you can have, a total of six. Both a wide-scope view and a narrow-scope view is available for each of the three different viewing styles. Table 6.1 lists the various possibilities.

TABLE 6.1 Browser viewing windows

| Viewing Style | Wide Scope View | Narrow Scope View |
|---|---|---|
| comprehensive | catalog | symbol |
| hierarchy | multi-class hierarchy | single-class hierarchy |
| class | multi-class browser | single-class browser |

Each wide-scope view encompasses the entire database. You access the three wide-scope windows through the CodeWarrior Window menu. Figure 6.1 illustrates these commands.

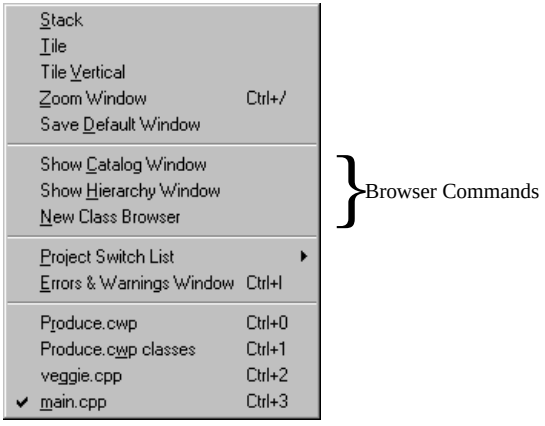



FIGURE 6.1
The CodeWarrior Window menu has commands to open wide-scope browser views.

Once you have a wide-angle look at the data, you can then focus on a smaller part of the data. This approach is rather like a zoom lens. You start out with the widest possible angle, identify an area of interest, and then zoom in on what you want to see close-up.

Each of the three principal viewing styles—catalog, hierarchy, and class—excels in answering certain kinds of questions. As you begin to master the browser, you develop a knack for knowing which kind of

view makes the best starting point for your work. However, you can always move quickly and easily among any of these viewing options. The browser has several simple and intuitive mechanisms for switching from one view to another.

While the browser and its multiple views might seem overly complex at this point, don't be alarmed. To a large extent all the windows work the same way. You do not have to master six completely different interfaces. The real question you want answered is, "How do I use the browser?" Let's start answering that right away.

## Activating the Browser

As always, first on the list of tasks is turning the feature on. To turn the browser on, you change a project setting. Choose the Project Settings item in the Edit menu, and then select the Build Extras settings panel, as shown in Figure 6.2.

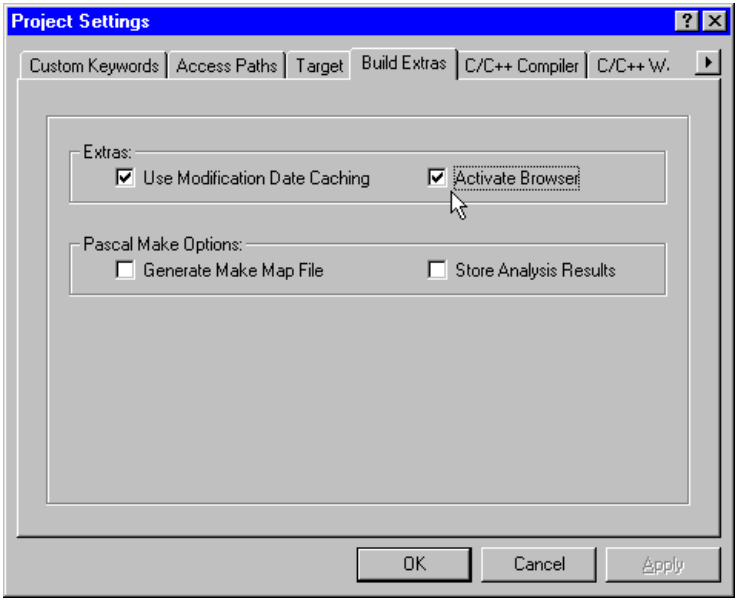
Click the Activate Browser check box to turn the browser on or off. This is a project setting, so it applies to the individual project. You can turn the browser on or off for each project. The state of the browser setting is saved and restored when you close and reopen a project.

That's all there is to it. When this option is on, the compiler automatically generates the browser's catalog every time you update your code.

Note one subtlety that derives from the close integration of the browser and compiler: The catalog depends upon the compiler. If you change files, the catalog is not updated automatically. You must recompile files to update the browser database. Similarly, if you turn on the browser after compiling files, you must recompile your files to generate the catalog of information.

FIGURE 6.2

The Build Extras settings panel includes the option for turning on the browser.



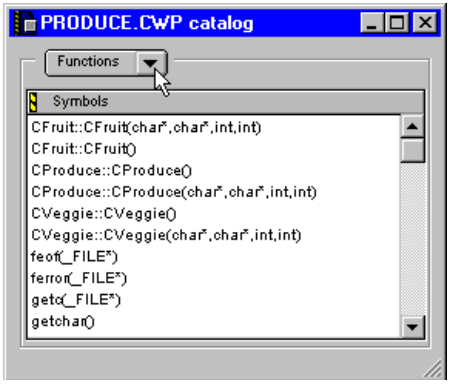

CodeWarrior saves the catalog in a separate file alongside the project. It has the same name as your project, with the added naming extension .cat.

After you have turned the browser on and compiled a project, you have a catalog of information available to you. As we did with the discussion of search and replace, let's start with the simplest ways of accessing data and move on toward more complex views and tasks.

## Accessing the Catalog

The catalog is the simplest view of the database. To open a catalog, choose the Show Catalog Window item from the Window menu, and the catalog window appears. Figure 6.3 shows you what the catalog window looks like.

FIGURE 6.3
The catalog window displays all instances of one type of symbol.




As I mentioned before, the browser identifies eight categories of data, as follows:

- Classes
- Constants
- Enumerated constants
- Functions
- Globals
- Macros
- Templates
- Typedefs

The catalog window shows one kind of data at a time. You decide which kind to view using the pop-up menu at the top of the window. All the symbols in the database that belong to the selected category appear in the catalog window in alphabetical order. Class member functions (methods) are alphabetized by function name, not class name, although the class name does appear in the listing.

**TIP** For C code, struct names appear in the classes category.

You can select one symbol at a time in the scrolling list. Use standard selection techniques—arrow key, or click. When you double-click an item in the list (or select it and press the Enter key), the editor opens the source file and scrolls to the source code related to that symbol. Exactly what gets displayed depends upon the nature of the symbol. Table 6.2 lists each category of data, and what code you see when you double-click a symbol.

TABLE 6.2 Accessing code from the catalog window

| Double-Click This Symbol | Editor Displays |
|---|---|
| class | class or struct declaration |
| constant | constant definition (value) |
| enum | enum definition (value) |
| function | function definition |
| global | global definition (value) |
| macro | macro definition |
| template | template class declaration |
| typedef | typedef declaration |

As you can see from Table 6.2, by double-clicking a symbol in the catalog window you see either the declaration or definition of the symbol, as appropriate.

**NOTE**

Note the distinction between the terms *declaration* and *definition* in code. A declaration tells the compiler what a term means. Strictly speaking, the source text in a declaration does not generate code. A definition, on the other hand, is source text that results in code generation. For example, a function prototype is a declaration. It tells the compiler what the name, return type, and parameters are for a function, but it does not actually generate any code. The actual code for a function—the function implementation—is the function definition.

The catalog window is ideal for simple questions such as, "What is the value of this constant?" With the catalog window you can instantly go directly to the source. You don't have to remember what file contains the definition, or where in the file the definition is located.

The catalog window is also ideal for locating a function definition in procedural code, or going to a class or struct declaration. Once again, you go right to the source code without having to worry about file navigation.

The catalog window is useful for all kinds of code, not just object-oriented code. It is an excellent navigational aid, and a welcome addition to your arsenal of programming tools. However, if you work with object-oriented code, the browser gets much better.

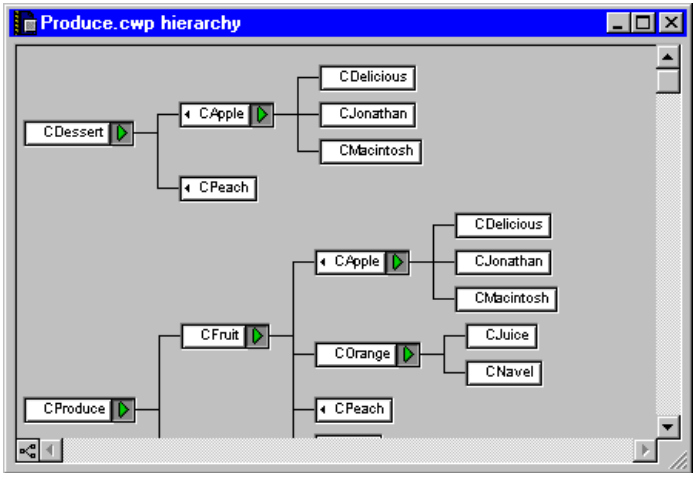
## Using the Graphical Browser

The CodeWarrior browser includes a graphical view of the inheritance relationships between classes in the database. This hierarchy view is an especially eye-catching and useful tool. You display the multi-class hierarchy window when you choose the Show Hierarchy command from the Window menu. Figure 6.4 shows you the multi-class version of the graphical browser.

Remember that the browser starts you off with a wide-angle view of the territory. The multi-class hierarchy window includes all the classes in the database. It is quite likely that they won't all fit in the window at once, so you can scroll around to see other parts of the hierarchy. You can also type ahead. This means that as you type a class name, the browser selects the class that fits what you have typed so far, and scrolls to display the class. You can also use the Tab key to cycle through the classes alphabetically.

Pure base classes (specifically, those that have no ancestors) appear at the far left. Descendants appear to the right. Lines join each descendant to its ancestor. I'll call any class with descendants a *base class*, and I'll call the descendants *subclasses*.

FIGURE 6.4
The multi-class browser window displays the class hierarchy for an entire catalog.




**NOTE**

A particular class can be both a base class to some subclasses, and a subclass to some other base class.

Classes that descend from a common ancestor, and that exist at the same level, are called *sibling classes*. An entire set of classes based on a single pure base class is called a class *hierarchy*. Figure 6.5 illustrates the terminology and relationships.

## Controlling the Appearance of the Hierarchy Window

The browser draws lines between classes related by inheritance. You can draw those lines so that they are at right angles, as shown in Figure 6.5. Or, if you prefer, you can have the browser draw diagonal lines, as shown in Figure 6.6. You can find the control at the bottom left of the window. Click this button to switch between right-angle lines and diagonal lines. This feature is cosmetic only. Use whichever style appeals to your aesthetic sense.

FIGURE 6.5

A class hierarchy displays the various relationships among classes.

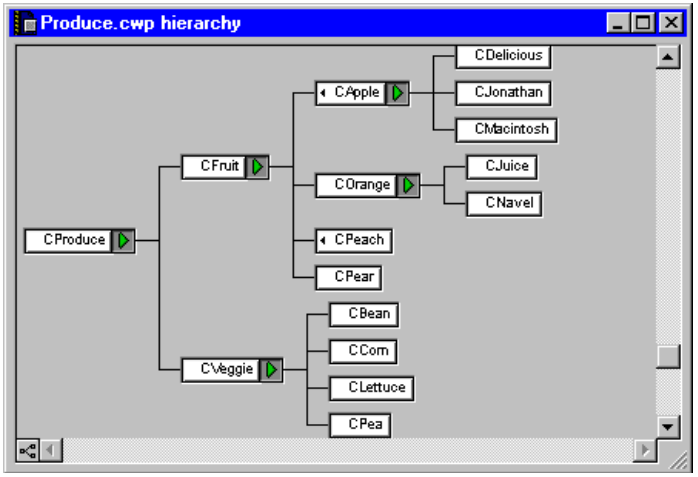
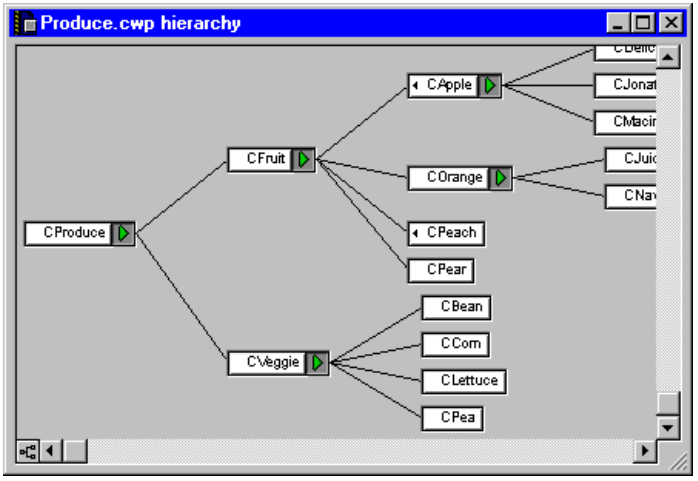



FIGURE 6.6

The browser can display connections between classes with diagonal lines.



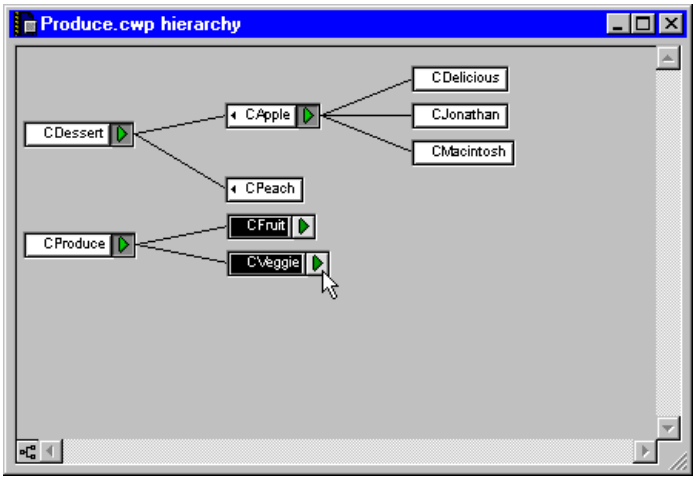

## Showing and Hiding Classes

As you can see from Figures 6.6 and 6.7, each class has a box that contains the class name. To the right of each name is a triangle. This is a

disclosure triangle. Click this triangle to show or hide subclasses. Figure 6.7 shows a class with no subclasses shown. After a click, the subclasses appear.

FIGURE 6.7

The disclosure triangle for a base class opens or closes that branch of a class hierarchy.

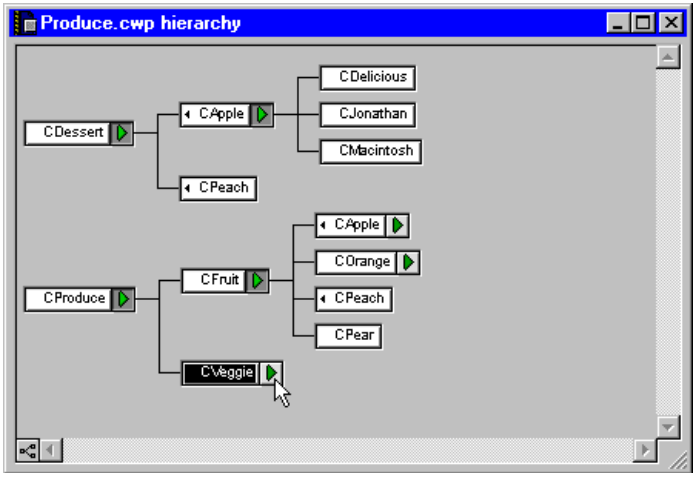



Precisely how this works is worth a little explanation, because CodeWarrior gives you real power over showing and hiding classes in any or all class hierarchies on display.

A class hierarchy can be arbitrarily complex. You may have several levels with all sorts of branches. Every base class has a branch of classes below it. At any moment, all, some, or none of the subclasses on any given branch may be visible. Let's call this the branch's visible *state*. For example, if you had every subclass visible in a five-level class hierarchy, you could turn some off at level four. Figure 6.8 shows a hierarchy with parts of the class hierarchy hidden.

Now, what happens when you click the disclosure triangle for the pure base class at the far left? The branch from that class down is hidden. That's good.

FIGURE 6.8
A partially concealed class hierarchy represents the visible state of this hierarchy.




What happens when you click the same class disclosure triangle again? The branch reappears in exactly the same state it was when you hid it. Any subclasses that were hidden when you closed the branch are still concealed when the branch reopens. Clicking the triangle does not display *all* subclasses. Clicking either hides a branch or shows a branch. When the branch appears, it appears in the same state it had when you hid it.

This is exactly what you want to happen. You may have a very complex hierarchy, and you intentionally conceal parts of it to keep the view manageable. CodeWarrior respects your wishes and remembers the state of each branch in the hierarchy.

**TIP**

You can select a class in the hierarchy by typing its name, but type-ahead selection applies only to classes currently visible in the hierarchy.

## Performing Deep and Wide Disclosures

A class hierarchy can run quite deep, with several levels of classes, and several branches. A click always opens all direct subclasses but beyond that the click may or may not disclose other subclasses. It all depends on what was visible the last time you looked. Sometimes you want to see an entire hierarchy, and it would certainly be unpleasant if you had to click all the disclosure triangles in a hierarchy to see the whole thing.

You can display all levels of subclasses (that is, disclose an entire branch) if you Alt+click the triangle. If you do this for a pure base class you can display or hide that entire class hierarchy with a single click. Let's call this a *deep* disclosure.

**NOTE**

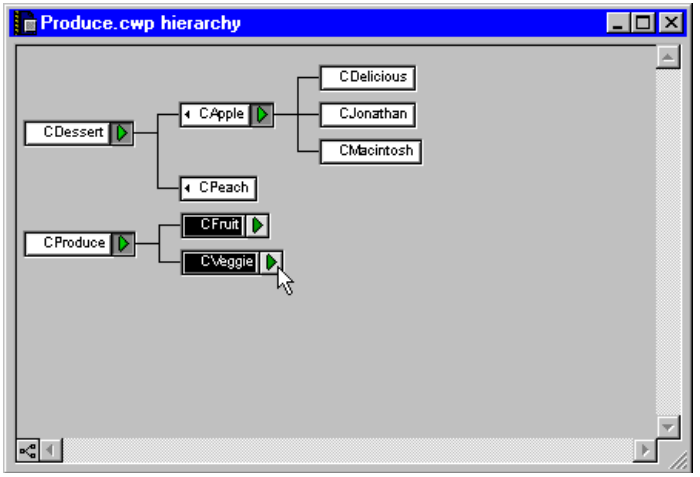
If you Alt+click to hide a branch, the browser "forgets" the previous state of the branch. Next time you do a simple click to expose the branch, you will see the next level down, and no further.

You can also work with all siblings. Figure 6.9 shows a class hierarchy where some sibling classes each have subclasses. You can show or hide all branches for all sibling classes with a Ctrl+click on any sibling-class disclosure triangle. Let's call this a *wide* disclosure.

A wide disclosure is the same as if you manually clicked on each sibling class disclosure triangle. Branches for each are restored to their previous visible state. This procedure does not do a deep disclosure.

However, you can combine the two procedures and do a *deep and wide* disclosure. Alt+Ctrl+click on the disclosure triangle for a class, and all the subclasses for all sibling classes appear or disappear.

FIGURE 6.9
Ctrl+click a triangle to display or hide the branches for all sibling classes.




**TIP**

Although pure base classes are technically not sibling classes, if you Alt+Ctrl+click a pure base class, you do a wide and deep disclosure for the entire database of classes. Use this trick to show or hide every class hierarchy.

## Seeing Multiple Inheritance

In C++ a class may have multiple ancestors. The ability to see not just the principal base class, but *every* base class is vital to your complete understanding of a class hierarchy.

In the multi-class hierarchy view, any class that has multiple ancestors has a small black triangle to the *left* of the class name. If you click on that triangle, a small pop-up menu appears listing each base class. Figure 6.10 shows such a menu.

If you select one of the items off the menu, the browser window automatically scrolls to that class. This response lets you navigate instantly to *any* base class, not just the principal base class.

A particularly adept C++ programmer familiar with the intricacies

FIGURE 6.10

A class with multiple ancestors has a menu that lists each base class.

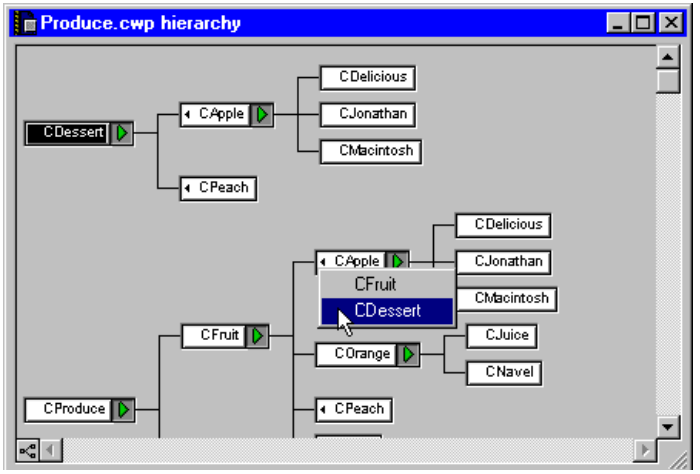
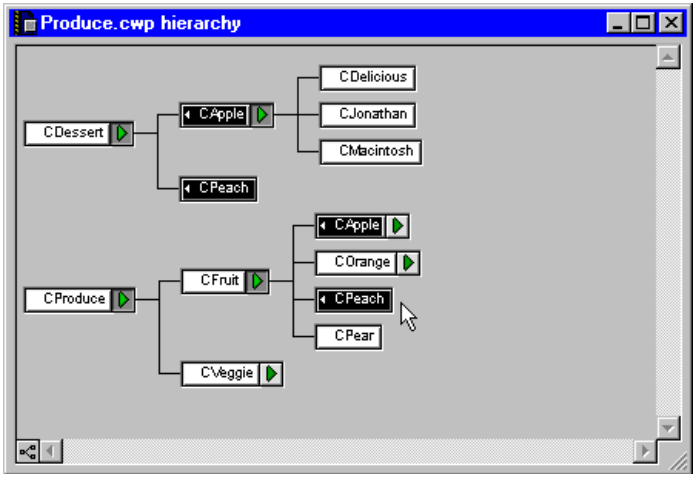



FIGURE 6.11

Classes with multiple ancestors appear in multiple hierarchies.




browser handle the fact that a class with multiple ancestors belongs to two or more separate and distinct class hierarchies?

The answer is simple. The class appears in both. Figure 6.11 shows two separate hierarchies. Here, both the CDessert and CProduce hierarchies have the CApple and CPeach classes in them.

This duplication doesn't cause any problem. Remember, the browser is simply a specialized database viewer. Both entries for the class point to the same data in the code. You can get at the real data using any entry in the multi-class hierarchy window.

Finally, this discussion brings us to the big question: "What do you use this view for?"

## Navigating Code with the Hierarchy View

It is clear that you have great control over what parts of the class database you want to see. Just the fact that you can see the inheritance chain simply and clearly is a great help. It is easy to see which classes descend from other classes. All you have to do is look. You can't see these kinds of relationships by looking at source code.

The first and primary purpose of the browser's multi-class hierarchy view is what I have already talked about—seeing the relationships among your classes. You can look up or down an inheritance chain, and across to siblings and multiple ancestors.

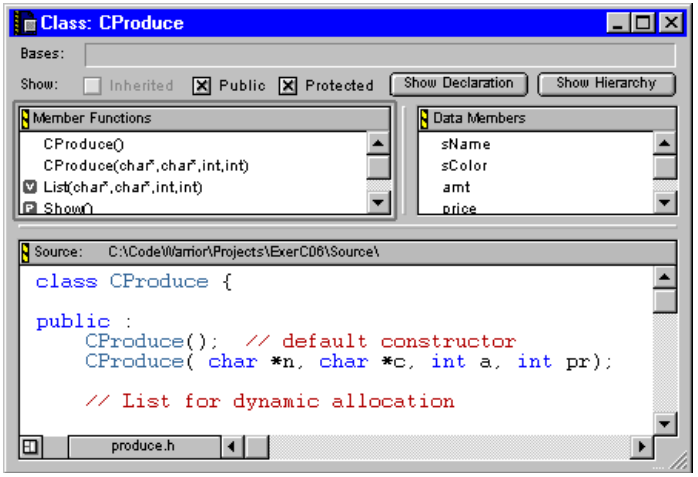
You can also use the hierarchy window as a code navigator. If you Alt+double-click a class name, the class declaration appears in an editor window. This is the same as if you had double-clicked the class name in the catalog window.

The multi-class hierarchy window also serves as a gateway to seeing your code at a finer level of detail. A simple double-click on a class name opens up a single-class browser for that class. We'll discuss class browsers in a bit. However, Figure 6.12 gives you a quick preview. As you can see, the class browser lets you identify any class member (function or data), and work with the related source code.

Finally, here's the neatest innovation of the class hierarchy view. If you click and hold the mouse button down on a class name, a marvel-

FIGURE 6.12

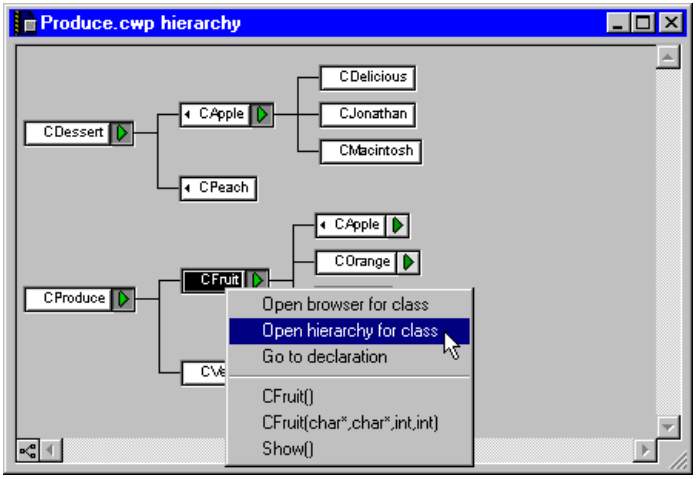
The single-class browser window lets you examine all the members of a class.




ous pop-up menu appears, shown in Figure 6.13. This is the context menu. You will meet this menu again.

FIGURE 6.13

The context pop-up menu lists all kinds of related places where you might want to go.



The context menu lists a variety of code locations (contexts) that are related to this class. The first three items let you open a single-class browser, a single-class hierarchy window, or open the class declaration in an editor window. The context menu also lists each and every function actually defined in the class. If you select a function name, you go to the source code for that function. Very nice.

**TIP**

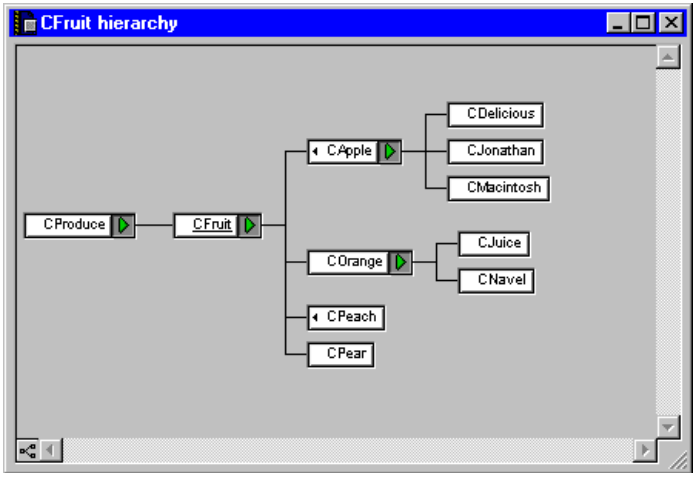
Here's what might be the coolest tip in all of CodeWarrior. When the browser is on, you can see a context menu for *any* symbol in *any* editable view. This includes editor windows, the search results window, and any other IDE text view where you can edit text. Just click and hold the mouse button on a symbol, and the context menu appears. The contents of the menu vary depending upon the nature of the symbol. However, the menu always lists the most likely places where you would want to go to see related code. This is an absolutely fantastic little tool. The context menu alone is enough of a reason to leave the browser on all the time for all your code.

## Viewing a Single-Class Hierarchy

As you refine your search for information, you may want to focus on the inheritance relationships of a single class. The principal way you open a single-class hierarchy window is through the context menu I just discussed. Click and hold the mouse button down on the name of a class, and then choose the correct item to open a single-class hierarchy window for that class. When you do, a window like that shown in Figure 6.14 appears.

This window is identical in appearance and functionality to the multi-class hierarchy window. However, the scope of the data displayed in this window is much narrower. A single-class hierarchy window focuses on one class, and that class appears centrally in the window with its name underlined. Let's call this the *central class*. You cannot display any classes in this window except those related to the central class.

FIGURE 6.14
The single-class hierarchy window focuses on one class.




To the left of the central class is the immediate base class (or classes if the central class inherits from multiple base classes). However, further ancestors—those that are not the immediate base classes—do not appear. The display goes only one step back up the hierarchy.

To the right of the central class is its branch descendants. The disclosure triangle works just like it does in the multi-class hierarchy window. You can display the entire branch of subclasses, no matter how deep.

Everything else about this browser view is identical to the multi-class hierarchy. You can use the context pop-up menu, open and close the central class branch of the hierarchy, double-click a name to open a class browser, and so forth.

Let's look at the class browser now.

## Using the Class Browser

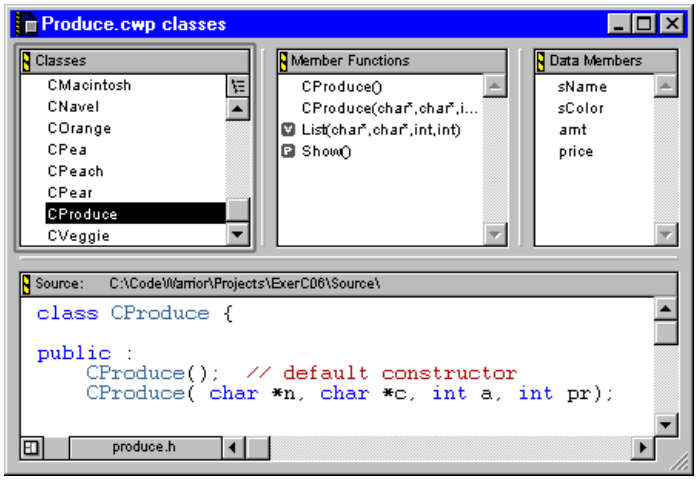
The CodeWarrior class browser is the principal mechanism through which you view class-based data in the catalog. The multi-class

browser is the wide-angle view that includes all the classes in your database. The single-class browser, like the single-class hierarchy window, concentrates on a single class.

When you choose the New Class Browser item in the Window menu, a multi-class browser window appears, as shown in Figure 6.15.

FIGURE 6.15

The multi-class browser window lists all the classes in a catalog.




You might have noticed that the other two menu commands that open views into the browser catalog, Show Catalog Window and Show Hierarchy Window, are worded differently. They say "show," not "new." That's not an inconsistency. You can have only one catalog window, and only one multi-class hierarchy window. By contrast, you can have several multi-class browsers if you want. If you have the screen real-estate available, you can have two or more class browsers open simultaneously. You can view different parts of your code in the different browsers, copy and paste text between them, and so on.

As usual, let's take a quick tour of the window before we go on toward the tasks you can accomplish using the multi-class browser.

## Understanding the Class Browser

The multi-class browser is a complex window, but it is really a collection of four panes that all work more or less the same, so mastering the multi-class browser is pretty simple. Let's look at how to work with panes in general, and then how to use each pane.

### Working with Multiple Panes

Four text-based panes appear in the window, but only one pane can be active at a time. The active pane has a gray focus box around it. When a pane is active, you can work in that pane. It receives all the keystrokes. You can change focus by clicking in the pane you want to become active, or by using the Tab key to cycle through the panes.

**WARNING**

You may recall from our discussion of the Search Results window in the previous chapter that you cannot cycle out of an editable text pane with the Tab key. The Source pane in the multi-class browser is an editable text pane, so be careful using the Tab key.

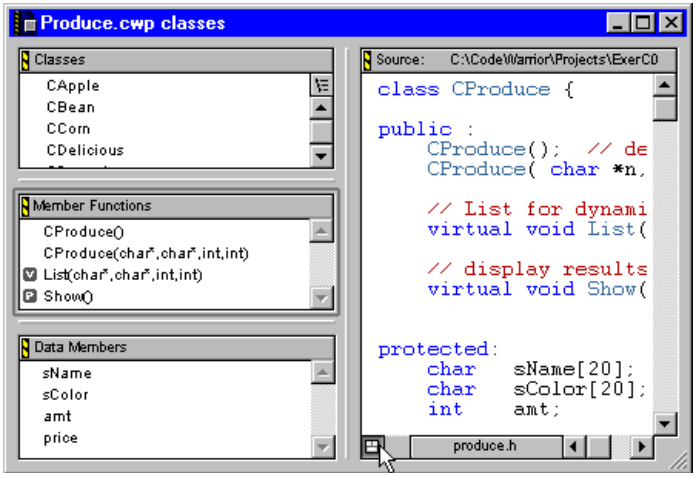
You'll find a pane resize bar between each pair of panes, so you can rearrange the space allocated for each of them as you wish. You can also arrange the three small panes vertically along the left side of the window, instead of horizontally across the top. Click the pane orientation button at the bottom left of the window. Figure 6.16 shows the multi-class browser in vertical orientation.

After you set up the window the way you like, you can choose the Save Default Window item in the Window menu to make this your default multi-class browser.

The three smaller panes, the Classes, Functions, and Data Members panes, are typical CodeWarrior scrolling lists. When the list is active, you can navigate through the list using standard techniques. You can

FIGURE 6.16

Toggle between vertical and horizontal orientation by clicking the pane orientation button, indicated by the arrow pointer.




click an entry to select it, use the arrow keys to move up and down the list, or type the name. As you select an item in the active pane, the source code on display in the Source pane changes.

**TIP**

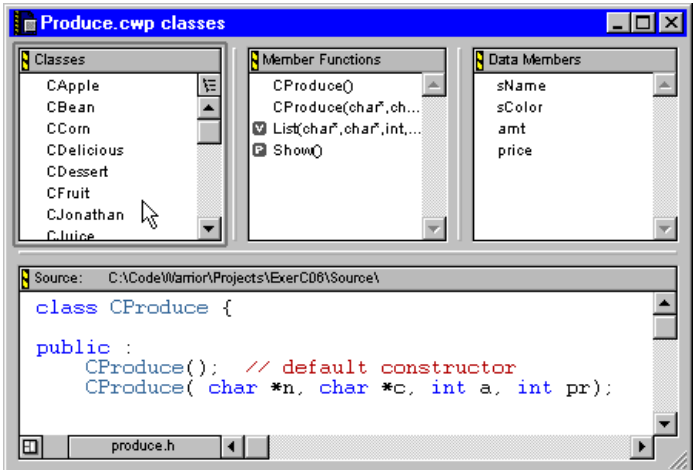
If you look closely you'll see that the orientation button in Figure 6.16 differs slightly from the one in Figure 6.15. In CodeWarrior, a button that toggles between two states is designed to show you what happens when you click the button. In this case, when you are in horizontal mode, the button shows vertical orientation because this is now the button to switch to vertical orientation. When you are in vertical mode, the button shows horizontal orientation. This design subtlety shows how carefully the CodeWarrior engineers considered the human interface in the IDE.

## Understanding the Classes Pane

The Classes pane at the top left of the multi-class browser window lists every class in the catalog. You can use this listing to decide which class you want to examine as you narrow your focus. Figure 6.17 shows you this pane.

FIGURE 6.17

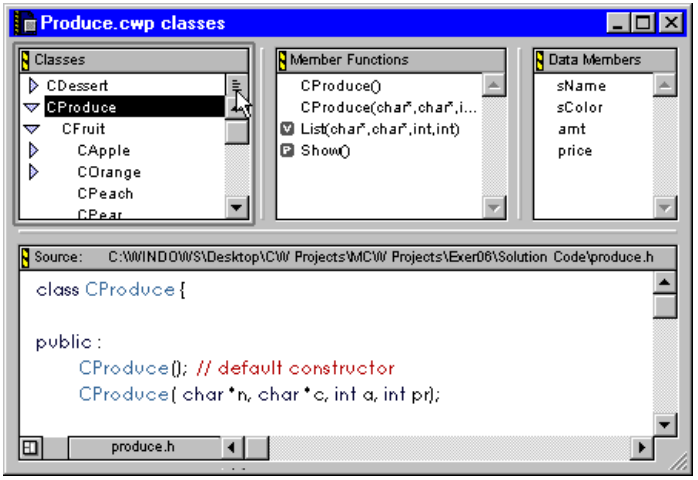
The Classes pane lists each class in the code database.




The sorting button controls how the classes are listed in this pane. You can switch between an alphabetical listing and a listing based on class hierarchy. Figure 6.18 shows the hierarchical listing.

FIGURE 6.18

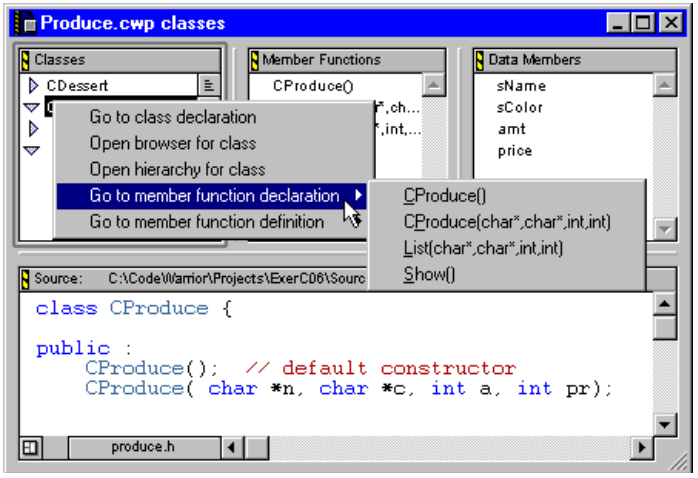
Toggle between alphabetical and hierarchical view by clicking the sorting button indicated by the arrow pointer.



When you select a class, the class declaration appears in the Source pane. In addition, you can access the context menu for a class. The context menu has several useful destinations, as shown in Figure 6.19. For example, you can jump to the class declaration in an editor window if you want, although you can view the same data in the Source pane right in the browser. You can also open up a single-class hierarchy window or single-class browser for this window. And you can jump to the declaration or definition of every member function actually defined in the class. If you choose one of these last destinations, the code will appear in an editor window.

FIGURE 6.19

You can go to useful destinations with the class context menu.




## Understanding the Functions Pane

The Functions pane, located between the Classes pane and the Data Members pane, lists every function actually defined in the selected class in alphabetical order. This pane lets you decide which function you'd like to examine in the Source pane. Functions that are inherited from an ancestor, but which are not defined in the class itself, are not listed.

Next to the function name there may be a small icon. The icon tells you whether the function is static, virtual, or pure virtual. Table 6.3 shows you the possible icons and what they mean.

TABLE 6.3 Function specifier icons

| Icon | The Function Is . . . |
|---|---|
| | static |
| | virtual |
| | pure virtual |

## The Fundamentals of Function Modifiers

A static function is like a global variable—it is always available. You don't need to have an object of the class to call the function. You can call a static function even if no object of this class is available.

A virtual function is one that you can override in a subclass. When a subclass inherits a virtual function, it can create its own version (an override) of that function, effectively replacing the inherited version. (Actually, the subclass then has two versions of the function, because it still has access to the inherited version as well.)

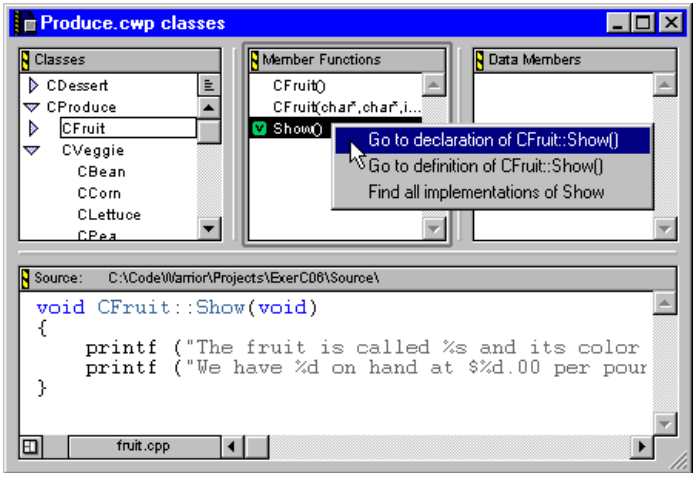
A pure virtual function is one that you *must* override in a subclass. Although the name and parameter list of the function is declared in the ancestor class, no function actually exists in the ancestor class. Therefore the subclass *must* create its own version of the function.

When you select a function in the Function pane, the definition appears in the Source pane.

The context menu for a function lets you jump to the function declaration, or, you can see all implementations of a function. Figure 6.20

FIGURE 6.20

The Function pane context menu lets you jump to the corresponding declaration.




shows the menu as it appears in the multi-class browser. (In other situations, a context menu for a function might have other options.)

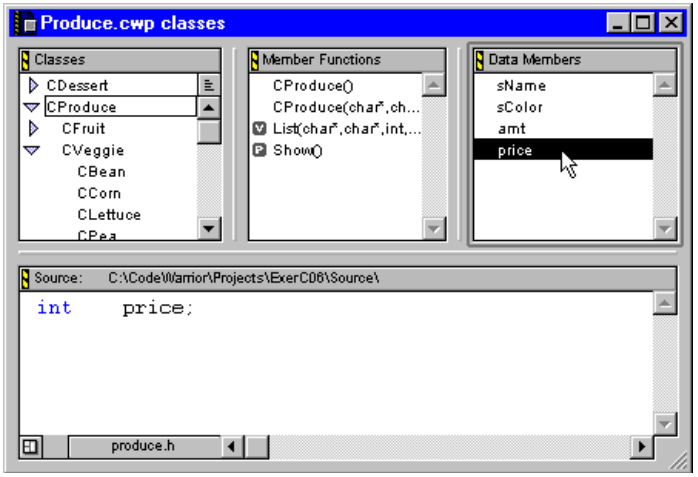
If you jump to see all implementations of a function, a Symbol window appears that lists every version of the function in all the classes in the project. We'll discuss the Symbol window in a little bit.

### Understanding the Data Members Pane

The Data Members pane lists every data member in the selected class alphabetically. You can use this list to see the data type for each data member quickly and easily. As previously explained in Table 6.3, static data members are denoted by the small "s" icons to the left of the names. Figure 6.21 shows the Data Members pane.

When you select a data member, the member declaration (which specifies the data type for the member) appears in the Source pane. The Source pane lists that line of code only. The context menu lets you jump to the same place, but in an editor window. You can then edit code around the declaration if you wish.

FIGURE 6.21
The Data Members pane lets you see all data members.




## Understanding the Source Pane

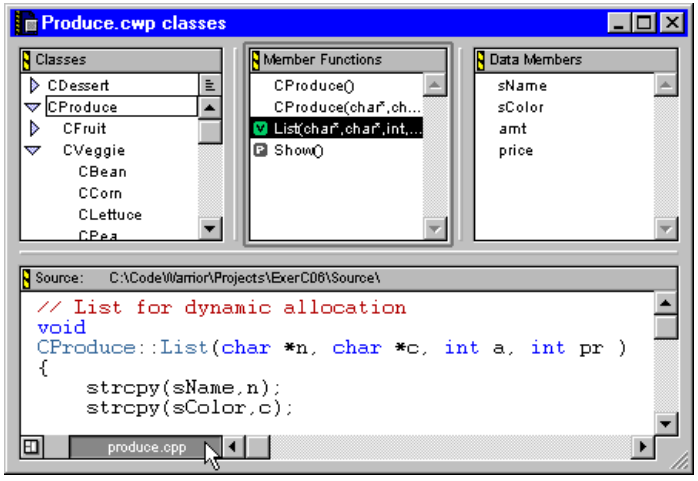
The Source pane displays class, function, or data member information depending upon what you have selected in the respective panes in the browser. The text on display in the Source pane is limited, because the browser excerpts the pertinent code from the original source file and displays it in the pane. You cannot see all the code in the file, only the code that is related to the class, function, or data member you are viewing.

In addition, the Source pane contains a couple of items worth noting, as shown in Figure 6.22. At the top of the pane, the path to the current file that contains the code on display is shown. At the bottom of the pane, the file button opens the file. As discussed before, the orientation button changes the arrangement of the browser window between horizontal and vertical.

Although the text in the Source pane is limited, you can edit the text directly. If you want to see more code, click the file button at the bottom of the pane, and the editor opens the file containing the code on display in the Source pane. You can then work on any code in the whole file.

FIGURE 6.22

The Source pane controls let you open a file in the editor.




You can also use the context menu from the Source pane. The context pop-up feature is fully functional for any symbol that appears in the Source pane.

## Viewing and Editing Code with the Class Browser

Of course, if you want to see code, all you have to do is select the correct class, function, and/or data member in the browser window. The related code appears immediately in the Source pane. At that time, all the usual techniques for editing code and saving your work, which you have already mastered, can be put into play.

But wait. A couple of more features can make entering a function call or other symbol into your code instantaneous.

If you're editing code in the Source pane, you can copy class names, function calls, or data members directly from the other panes in the browser. Just Alt+click the symbol you want to use, and that name appears in the Source pane at the current insertion point. If you're copying a function using this technique, the entire function prototype appears.

Using this trick provides you with significant advantages. First, it saves you typing, which can be great if you're using long names. Second, because the entire prototype appears for functions, you can't miss a parameter. You get one every time, and you just provide the local data to make a successful call. This strength can save you a lot of time because you avoid compiler errors when you build your code.

Although the code on display in the Source pane is limited to the immediate symbol selected in the browser, the same trick works in the editor window as well. If you use the File button at the bottom of the Source pane to display that same file in an editor window, and then Alt+click on a symbol in the Browser, that symbol appears wherever you are working in the file in the editor window, not just in the Source pane. This fact extends the usefulness of the Alt+click browser-editing trick beyond the immediate confines of the browser.

It gets even better, because you can do the same kind of thing at any time with any file! On the context menu for certain symbols, such as functions, you will often see an item that says "Insert template for" followed by a name. If you choose this item, CodeWarrior inserts the name of the function, along with all parameter data types, into your code at the current insertion point. After that, all you need to do is replace the parameter data types with variable names or values. Figure 6.23 shows this technique at work.

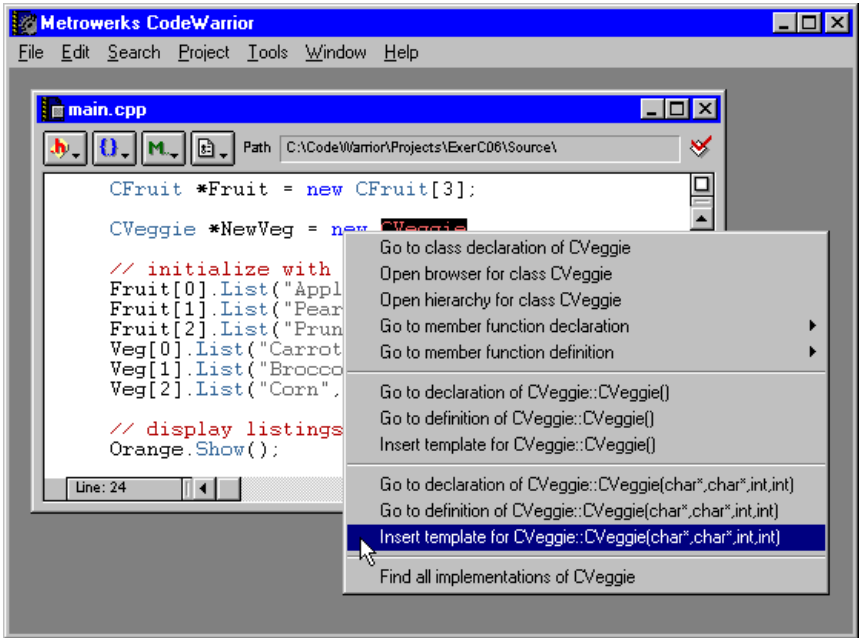
The browser does double duty both as an excellent navigation aid, and as a tremendous assist in actually writing code.

## Using the Single-Class Browser

The single class browser is very much like the multi-class browser, but with a narrower focus. You have several ways of opening a single-class browser, depending upon the view you are in when you decide you want to see code this way. You can double-click a class name in the hierarchy views, or use the context menu for a class name in the catalog or multi-class browser views.

FIGURE 6.23

Entering a function template into your code using the context menu.

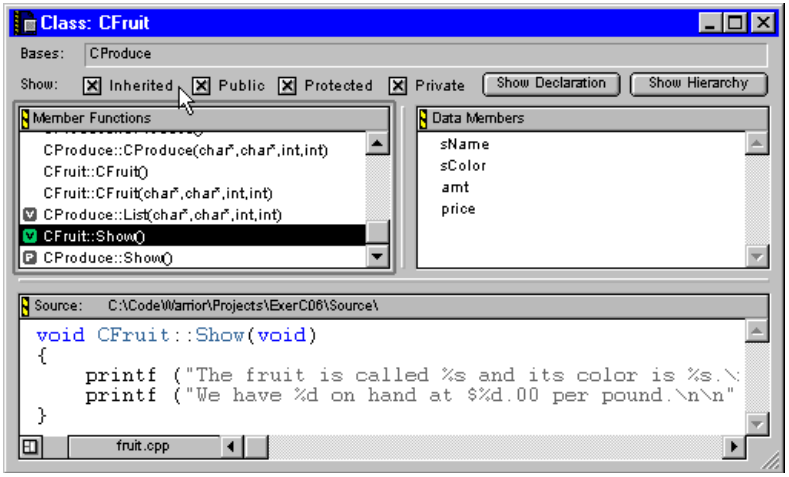



Whatever technique you use, when you open a single-class browser window, you'll see something like Figure 6.24.

As you can see, this window is very similar to the multi-class browser. Some differences exist, however. For one thing, the Classes pane is missing, because this browser focuses on a single class. The name of the class on display in this single-class browser appears in the window title. The top of the window lists each base class. This is a static display for informational purposes only. Beneath the base class list, you'll find several buttons.

The real difference between the single-class browser and the multi-class browser is that you can control or filter the data you see in the Functions and Data Members panes. Figure 6.24 shows the check boxes that control the filtering process.

FIGURE 6.24
The single-class browser window displays all the information for a single class.




The four options you have affect inherited, public, protected, and private members. If the check box for an option is on, then class members of that kind will be displayed in the browser. If the check box is off, the members will not appear. These options lets you filter exactly what you want to look at.

Pay particular attention to the inherited feature. It is perhaps the most important aspect of the single-class browser window. This is the *only* browser view where you can see inherited functions. That makes the single-class browser particularly powerful and useful. In every other browser view, when you want to see the classes in a function, only those classes actually defined in the class appear in the view. Code in inherited functions is not directly available. That's not true in the single-class browser.

If you turn on the inherited filter, every function from every base class that has been inherited by the focus class appears in the function list. When you select the class, the code for that class appears in the Source pane, regardless of the class it was defined in, or the file that contains the code. Once again, the artificial boundaries between files disappear.

This functionality can be invaluable when you are exploring a class. For example, you may override a function in such a way that your code does a little work, and then calls the inherited function. You might want to see what the inherited function does so you can fully understand what your function does. In the single-class browser, the code for the inherited function is just a click away.

The downside to viewing inherited functions is that there can be a *lot* of them—sometimes hundreds. The presence of so many functions can interfere with finding what you're looking for, especially when you're looking for a local function. That's why you can turn the inherited functions (and all the other categories of members) on and off at will with a simple click.

## Using the Symbol Window

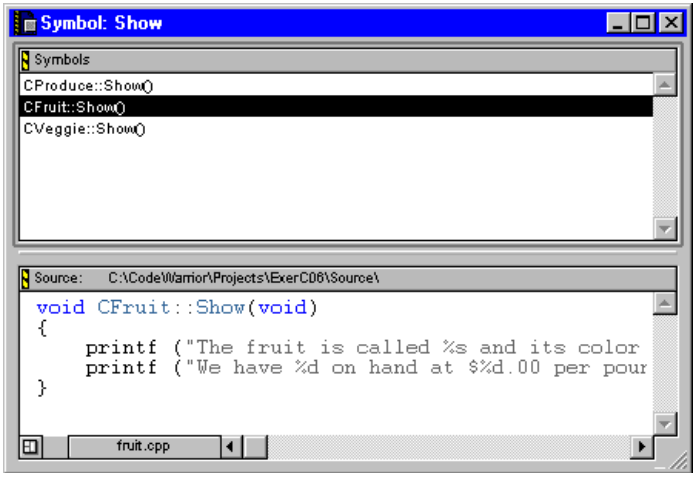
One more browser view requires discussion—the Symbol window, which displays a list of multiply-defined symbols. For example, you may have several versions of the same function scattered across a number of classes, in which case you can gather them all together in the Symbol window.

You get to the Symbol window through the context menu. Whenever you open a context menu, you will see an item that says something like "Find all implementations of" with a function name or symbol. If more than one definition exists for the item, CodeWarrior puts up the Symbol window, shown in Figure 6.25. If only one implementation exists, CodeWarrior opens an editor window for the file that contains the code and scrolls to that code.

This window should look pretty familiar to you, because you have seen all of its principal features in other browser windows. Everything works just like the single- and multi-class browser windows, including the orientation button.

FIGURE 6.25

The Symbol window displays lists of multiply-defined symbols.




The symbol on display appears in the pane title, and the pane at the top lists every implementation of that particular symbol. Simply select the version you want to examine, and the Source pane at the bottom displays the related code (although not the entire file). You can use the file button at the bottom of the Source pane to see the whole file.

**NOTE**

If you select a pure virtual function from the list, no source code appears in the Source pane because a pure virtual function has no definition.

The Symbol window is a great feature. It is analogous to the batch search results window you met in the last chapter and is one form of the CodeWarrior message window mechanism. In this case the list of items in the top pane comes from the browser database instead of a find operation, but other than that the functionality is essentially identical. You'll meet the message window again in the next chapter on compiling and linking, because CodeWarrior uses the message window to handle errors and warnings.

Although CodeWarrior can display multiply-defined symbols of any kind, the most obvious use for the Symbol window is to study and compare code in multiple functions. For object-oriented programmers, this feature is absolutely wonderful.

## Navigating Code with the Browser

The real purpose of the browser is code navigation. Once you find what you're looking for, then you can work with it.

You can use any browser view to open source files and work on code. In some views, such as the single- and multi-class browsers and the Symbol window you can edit code right in the browser. You already know everything you need to edit that code.

All that remains is using the browser to find the code you want. The good news: you know most of that as well. While discussing the browser interface, you have already met every major navigation technique.

A few things are worth pointing out, however, more or less as a summary of how to use the browser effectively. Let's look at how to move around between browser views, and at how to get to the three most common components you want to see: class headers, function declarations, and function definitions.

### Changing Browser Views

Now that you have met all the browser views, you probably realize that the CodeWarrior browser is an extraordinarily powerful and flexible tool. One fundamental aspect of getting around in your code is looking at it from the right perspective.

You should think about what you are looking for, and decide which approach—catalog, hierarchy, or class browser—is best. However, if you get it wrong, you haven't lost much time. You can move around

between browser views very easily. This flexibility is one of the nicest aspects of the entire browser architecture.

The catalog, multi-class hierarchy, and multi-class browser windows are always available via items in the Window menu. The Symbol window appears whenever you select the "Find all Implementations of" item in a context menu.

Table 6.4 shows you how to get from each view to the narrow-scope single-class hierarchy and browser windows. In each case, look for a class name in the original view. Then use the context menu, or double-click the class name as appropriate.

TABLE 6.4 Moving between browser views

| From Original View | To Single-Class Hierarchy | To Single-Class Browser |
|---|---|---|
| catalog | context menu | context menu |
| multi-class hierarchy | context menu | double-click |
| single-class hierarchy | n/a | double-click |
| multi-class browser | context menu | context menu |
| single-class browser | hierarchy button | n/a |

## Viewing Class Declarations

You'll often want to see a class declaration as you explore your code, and the browser lets you get to the class declaration in various ways from almost any browser view. For example:

- From the catalog window, first view the classes in the database, and then:
  - select a class and press the Enter key
  - double-click the name
  - use the context menu

- From the multi-class hierarchy window:
  - Alt+double-click the class name
  - use the context menu
- From the single-class hierarchy window:
  - Alt+double-click the class name
  - use the context menu
- From a multi-class browser window, you can:
  - select the class to see the declaration in the browser itself
  - use the file button to open the header file
- In a single-class browser:
  - use the Show Declaration button

You aren't likely to have multiply-defined classes, so the Symbol window doesn't apply here. Nevertheless, from each of the other five browser views you have one or more techniques for displaying a class declaration. You don't have to memorize them all, just use the methods that intuitively appeal to you. That's the beauty of CodeWarrior's well-designed interface.

## Viewing Functions

Of course, a class declaration isn't the only thing you want to see. The remarkable flexibility shown with class declarations repeats itself for function-related code. Once again, it's easy to see a function definition from any browser view. For example:

- From the catalog, first examine the functions, and then:
  - select a function and press the Enter key
  - double-click a function
  - use the context menu to get to either the definition or the declaration

- From the single- and multi-class hierarchy windows:
  - use the context menu to get to either the definition or the declaration
- From the single- and multi-class browser windows, first select the function in the Functions pane, and then:
  - the definition appears in the Source pane
  - use the file button to open the source file
  - use the context menu to get to either the definition or the declaration
- From the Symbol window displaying multiply-defined functions, select the function of interest, and then:
  - the definition appears in the Source pane
  - use the file button to open the source file
  - use the context menu to go to either the definition or the declaration

Many other kinds of symbols are available besides classes and functions, and you have many ways you can navigate in the browser. It is quite literally impossible to define all possible paths through this web of interlocked views into your code database.

However, this discussion of how you would navigate to three common pieces of code—class declarations, function declarations, and function definitions—should give you a flavor for how it all works, and how easy it is to get to precisely the right spot in your code. It doesn't matter how deeply the code is buried, or how completely you have forgotten your file organization. The browser delivers you to the right spot.

## Cool Features

The browser is really a collection of cool features. The context pop-up menu alone rates as one of the coolest things in all of CodeWarrior,

and we have already talked about it and a number of other cool features found in the browser. Only one amazingly great and cool feature of the browser remains to be discussed. If what you have seen isn't cool enough, this highlight should really impress you.

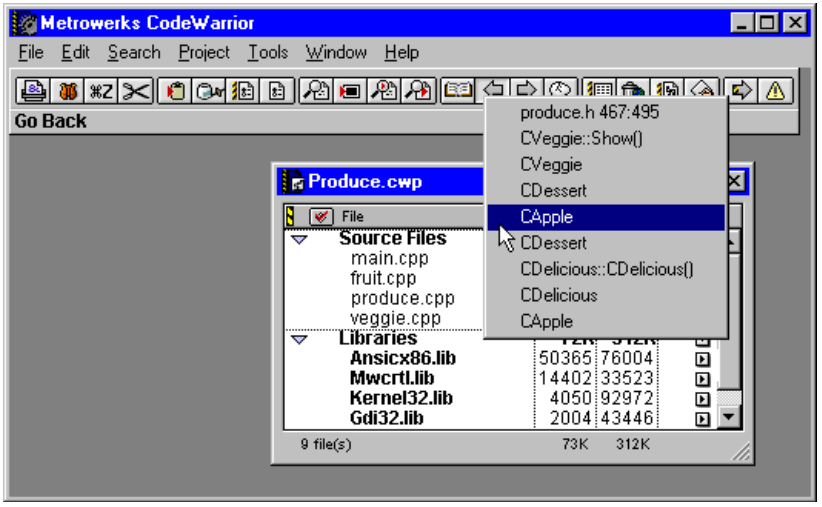
## Going Backward and Forward

The browser implements a kind of time machine that lets you return to browser views of the past. When the browser is active, it remembers everything you looked at using the browser. For example, let's say you use the catalog window to open up one file and make a change. Then you close the file, open a second file, and look at the code there. After that, open up a hierarchy window and use it to access some other file, which opens. You continue to work in your code, using the browser to open files and make changes.

Then you decide that you need to review something you saw in the murky past. You can choose the Go Back command in the Search menu to step back through every window and view you have used in the browser. If you have closed a window, it reopens and scrolls to the place you were looking at. Each step of the way back into the past, the browser restores the window you were looking at. It can be a source file, an include file, a hierarchy window, a class browser, the symbol window, and so forth.

The Go Back command is a quick way to go back to previous browser views. Once you are back in the past, you can use the Go Forward command to return to the present if you wish. If you use the toolbar, you can see the entire queue of saved views. Click and hold on the Go Back or Go Forward icons in the toolbar and a pop-up menu appears that lists every view you have been in. Select the place where you want to go and jump there instantly. Figure 6.26 shows you how it works. The view you are currently looking at is underlined in the view list.

FIGURE 6.26
Use the Go Back or Go Forward icons in the toolbar to return to other views.




The browser remembers up to 100 views. You can sail back and forth through these views at will. This is a very nice mechanism for getting back to a place when you *know* you've seen something, but you can't quite remember where it is. The browser remembers for you.

**TIP**

Previous changes you made are *not* undone as you go back to the past. All you are doing is taking a nostalgic look at a previous window opened by the browser. If you want to undo a series of actions, make sure you have the multiple undo feature turned on and use the Undo command in the Edit menu.

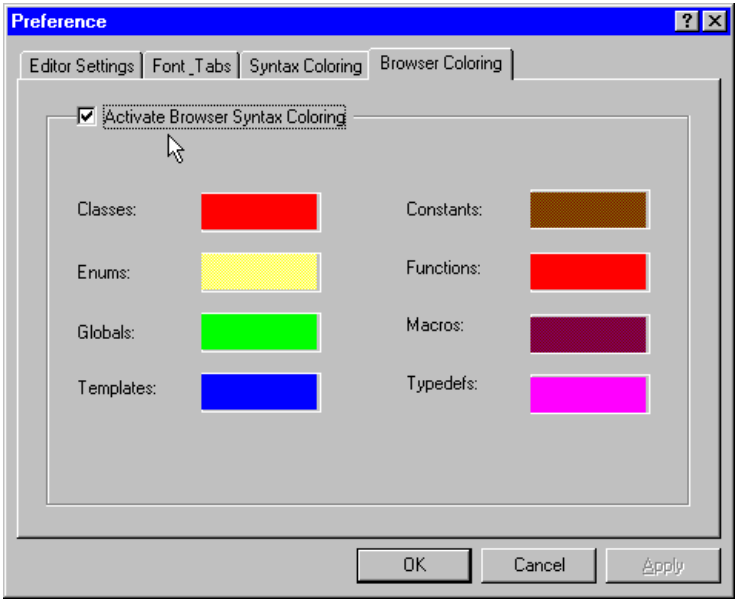
The browser viewing queue does not include windows you open via any other means besides the browser. If you open a file from the project window, or use the Open command from the File menu, those windows do not appear in the viewing queue. This list is strictly a browser feature, but it is a good one.

## Setting Browser Options

You're almost done. Before we leave the browser we should discuss a few settings you can modify that affect its appearance and behavior. You can specify the color used to display the eight different categories of data found in the browser catalog. You can also specify the length of time you must hold the mouse button down before the context menu appears.

To set browser colors, choose the Preferences item in the Edit menu, and open the Browser Coloring panel. Figure 6.27 shows this panel.

FIGURE 6.27
Use the Browser Coloring preferences panel to set display colors for symbols.

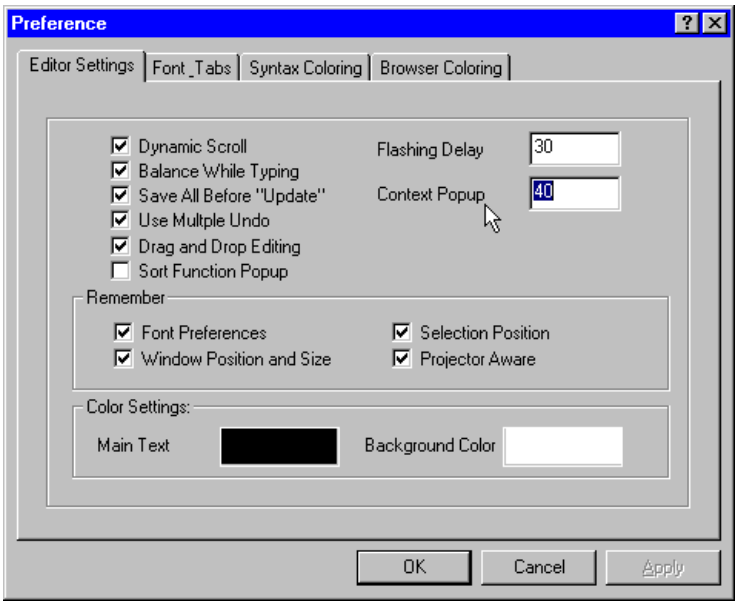



You can turn coloring on or off, and set individual colors for each category of symbol found in the browser database. This is a preference, so it applies to all your projects, but it has no effect unless the browser itself is on.

The other option you can control is the time delay before the context menu appears. This is a preference as well. Open the Editor Settings preference panel and look for the Context Popup delay item, as shown in Figure 6.28.

FIGURE 6.28

Set the delay time for the context menu in the Editor Settings preference panel.




The number in this box is in 60ths of a second. The acceptable range is from 0 to 240. A value of 30 or 40 seems to work well for most programmers, about half a second. A value of zero turns the context menu off, but if you are going to use the browser the context menu is simply too valuable to turn off. It is clearly the most powerful navigational tool in all of CodeWarrior. The context menu is available from almost anywhere—in every editable text view in CodeWarrior, and all over the browser.

You now have all the knowledge you need to master the CodeWarrior browser. There's only one thing required to become a true master of this tool—experience. So, let's get some.

## Hands-On Exercises

In this exercise you use the browser to explore some simple code in a project named Produce.cwp. The source files in this project describe a hierarchy of classes based on grocery-store produce—fruits and vegetables. You'll see two class hierarchies, one based on a CDessert class and a second based on CProduce.

Because the point of this exercise is to demonstrate the browser, and not object-oriented programming, many of the classes are essentially empty. The only functions in most classes are constructors, and even those aren't used. However, the examples include enough real code to demonstrate all the features of the browser fully and completely.

In this exercise you will turn on the browser, build the project, and then use the various features of the browser to navigate through the code in Produce.cwp. In the next chapter you'll actually build this code into a simple application.

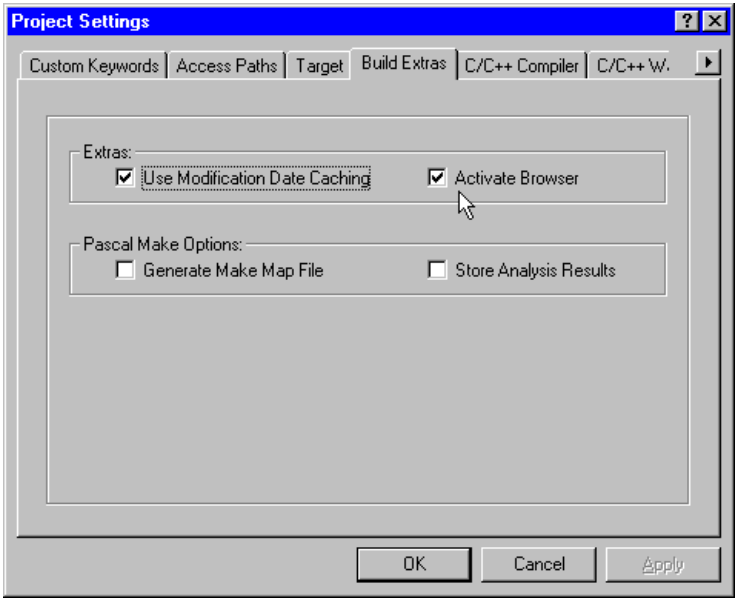
**1. Launch CodeWarrior and open a project.** If CodeWarrior is not already running, launch CodeWarrior and then open the Produce.cwp project file in the Start Code folder for this chapter, inside the Exer06 folder.

**2. Activate the browser and build the catalog.** Go to the project settings for this project (use the Project Settings item in the Edit menu), and click the Build Extras Tab. Turn on the browser by clicking the Activate Browser check box, as shown in Figure 6.29. Click OK to save this change and close the settings panel.

After turning on the browser, choose Bring Up To Date from the Project menu. Doing so compiles each file in the project. Because the

FIGURE 6.29

Turn on the browser so the compiler can generate the catalog.




browser has been turned on, the compiler will generate the database of information from the code it encounters during the compile. When the compiler is done, you're ready to work with the browser.

**TIP**

If you use the browser regularly you should have the browser on by default all the time so you don't have to remember to turn it on. You can do this by activating the browser when no project is open (thus setting the CodeWarrior default) or by using project stationery with the browser setting on.

**3. Use the catalog window.** In this step you use the catalog window to navigate to special places in your code. To begin, choose the Show Catalog Window item from the Window menu, and the catalog appears.

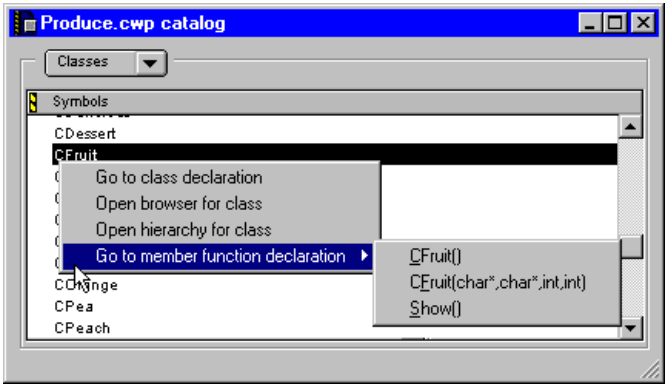
Examine each symbol type by clicking in the pop-up menu and selecting each type in turn. This simple project has no constants, glo-

bals, or templates, but you'll see plenty of examples in the catalog of the other types of data.

For example, view the classes in this catalog and then click and hold on the CFruit class name to see the context menu for the class, as shown in Figure 6.30. Use the menu to jump to someplace intriguing, such as the function declaration for **CFruit::Show()**.

FIGURE 6.30

Use a class context menu to open other browser views, or see the declaration or function definitions.

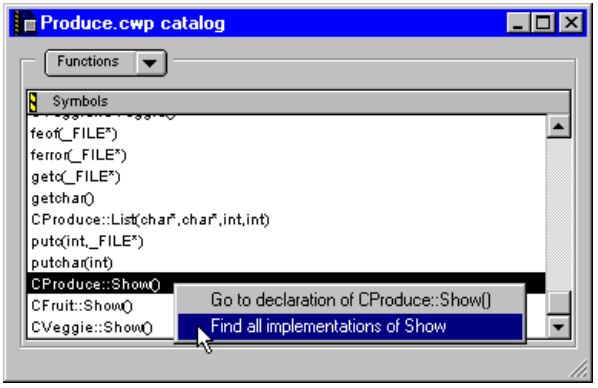



Return to the catalog window and look at the enums in this project. Double-click one to go to its declaration. In this project all the enums are in the standard library code, but you can still jump to that code.

Returning once again to the catalog window, switch to a display of functions and then click and hold on the **CProduce::Show()** function name to display the context menu for that function. Figure 6.31 shows you this process. Choose the Find All Implementations of Show item so you can see the symbol window that lists them.

Feel free to explore the symbol window if you wish, but a subsequent step covers it in detail. This step focuses on the catalog window.

FIGURE 6.31
Going to the symbol window from the catalog is quick and easy.




Continue to explore the catalog window to see all the ways you can use it as a starting point to move around in your code.

**TIP**

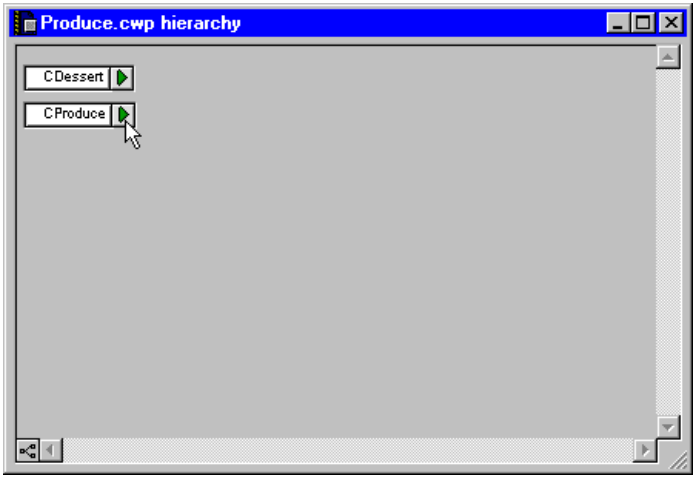
When you look at functions, notice that procedural functions are included in the catalog along with C++ class member functions. The catalog is good for all kinds of code, not just object-oriented code. Select one of these library functions, such as **getchar(),** and you can go right to the source.

**4. Use the multi-class hierarchy window.** Choose the Show Hierarchy Window item in the Window menu to display the multi-class hierarchy view. As with the previous step, the purpose of this step is to explore this view. You should experiment and play around with the view to get comfortable with how it works.

For example, click disclosure triangles to open and close branches of the class hierarchy. Use the Ctrl and Alt keys when clicking the triangles to see how they affect disclosure. One good test is to Ctrl+Alt+ click the disclosure triangle for CDessert and watch what happens. Do it again, and observe the results. One click hides everything except the two pure base classes, CDessert and CProduce, as shown in Figure 6.32. The other click displays every class in both hierarchies.

FIGURE 6.32

You can hide everything except pure base classes by Ctrl+Alt+clicking the disclosure triangle for any pure base class.




With the full class hierarchy visible, look for the CApple class (which uses multiple inheritance) and click on the little black triangle to the left of the name—a pop-up menu appears listing the ancestor classes. Choose one and it becomes the selected class in the view. In a small catalog such as this, selecting a class doesn't get you very far, but in a complex hierarchy this navigational aid is quite nice.

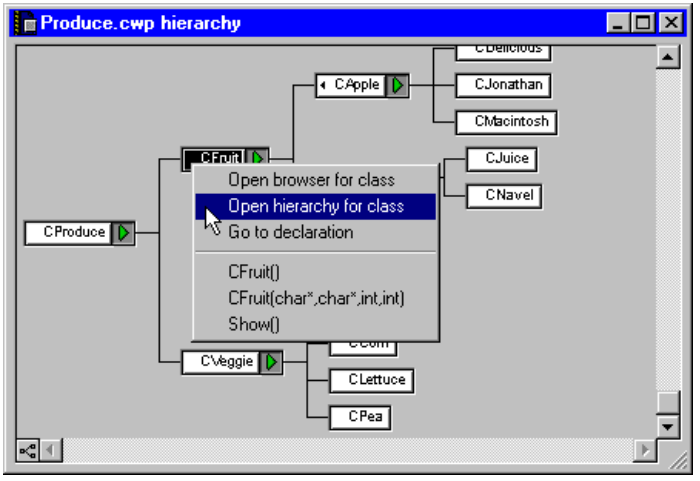
Double-click a class name and watch what happens—a single-class browser view opens giving you yet another avenue for exploration. Don't go too far afield, though, because you will meet the single-class browser view in a subsequent step.

Finally, display the context menu for the CFruit class and observe your choices—Figure 6.33 shows you what they are. Use this menu to open up a single-class hierarchy view for the CFruit class to get ready for the next step.

**5. Use the single-class hierarchy window.** Use this view to explore the single-class hierarchy, just as you used the multi-class hierarchy view in Step 4. In fact, you can repeat everything you did in the previous step—manipulate the display of the class hierarchy, open a single-class browser, and so on.

FIGURE 6.33

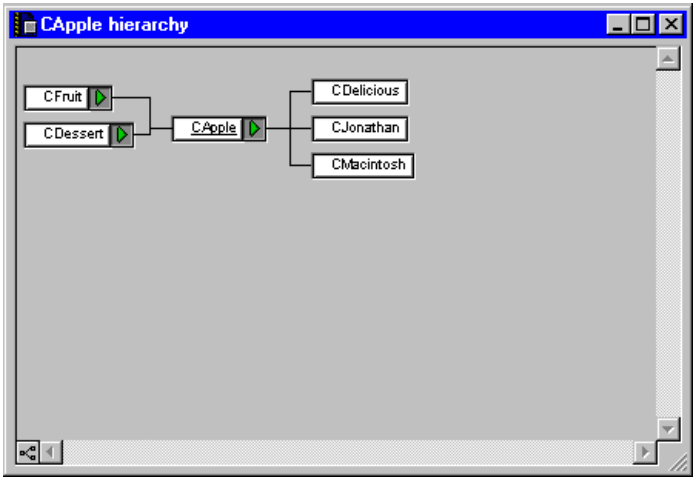
Use the class context menu to open a single-class hierarchy view for CFruit.




For example, locate the CApple class and then use the context menu to open a single-class hierarchy view for that class so you can see how it handles multiple inheritance. Both immediate ancestor classes are shown, as illustrated in Figure 6.34.

FIGURE 6.34

You can see all immediate ancestors in a single-class hierarchy view.



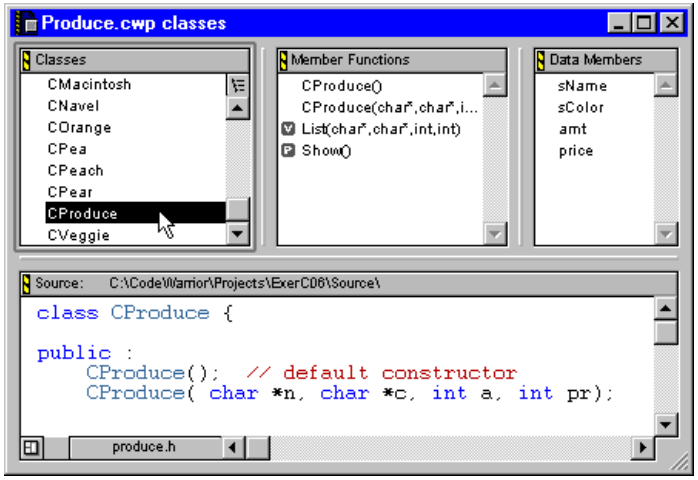
In every respect the single-class hierarchy works like the multi-class hierarchy except that you cannot display any classes in this window except those related to the central class.

If you find yourself with a profusion of windows at this point, feel free to close any or all of them (except the project window).

**6. Use the multi-class browser window.** As with all other wide-angle views of your data, an entry in the Window menu opens up this window—New Class Browser. Choose that item to see the classes in this catalog displayed in a traditional class browser, as shown in Figure 6.35. Select a class, a function, and a data member (the CProduce class has data members, so it's your best choice for this step) and observe what happens in the Source pane as you select each item.

FIGURE 6.35

The multi-class browser lets you navigate anywhere in your classes.



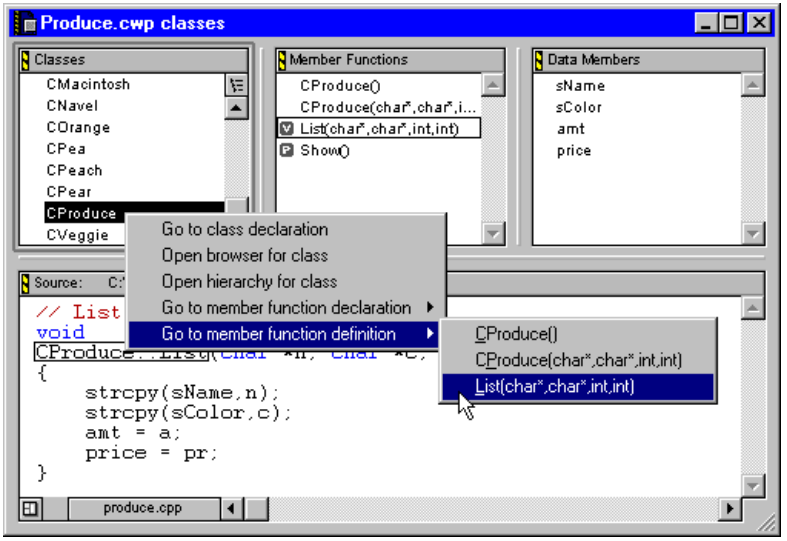

You can rearrange the parts of the browser. Use the button in the Classes pane to view the classes hierarchically or alphabetically. Click the pane orientation button (at the bottom left of the Source pane) to reorganize the panes vertically or horizontally. You can also resize the various panes by dragging the separator bar between the panes. Once

you have the window looking the way you like, choose the Save Default Window item from the Window menu to preserve your work.

Now you can use the window for its intended purpose—code navigation. The context menu works in all the panes, so give it a try. Display the context menu for the CProduce class, and observe where you can go—including to the function definition for any function in the class as shown in Figure 6.36. Choose the **List()** function and the source code file opens right at that function.

FIGURE 6.36

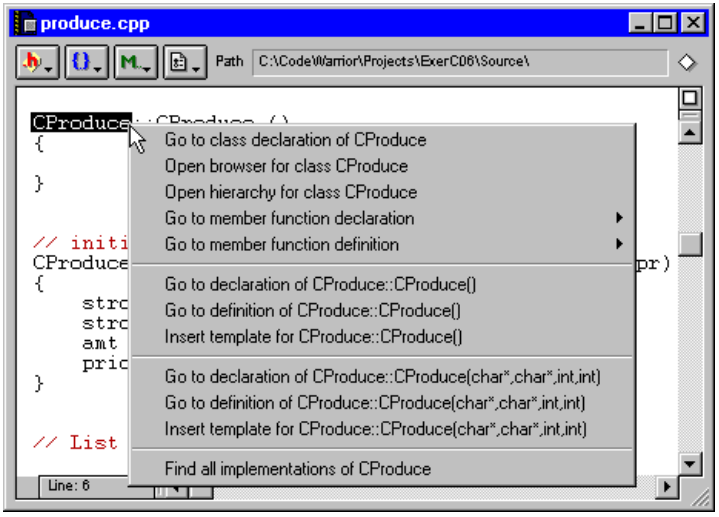
Once again, the context menu takes you where you want to go.




While the Produce.cpp file is open, display the context menu for the CProduce class right in the editor! Remember, this is a regular editor window but the browser's context menu still works. Click and hold on an instance of the class name CProduce to display the context menu. Again, you can go to a wide range of places—the class declaration, various browser views, every function declaration, every function definition, and more. Figure 6.37 shows you what it looks like. The context menu is a truly phenomenal navigation aid.

FIGURE 6.37

The context menu works inside the editor as well as the browser.



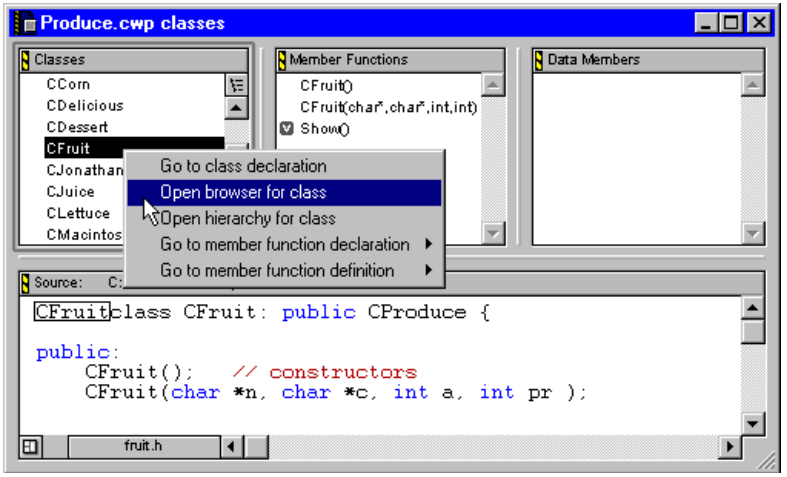

Return to the multi-class browser and select a function and then a data member. Observe what happens in the Source pane. Scroll in the Source pane to see what code is available to you for each kind of symbol, and you'll notice that the amount of code is limited. Use the file button at the bottom of the Source pane to open the associated file in an editor window.

Finally, to get ready for the next step, display the context menu for the CFruit class and open a single-class browser for it. Figure 6.38 shows one way you can accomplish this goal.

**7. Use the single class browser window.** When the single-class browser appears, take a look at it. Notice the check box "filters" along the top. For this step, click the Inherited check box so you can see inherited members as well as those defined in the CFruit class. Figure 6.39 shows you what it looks like. Once you have the inherited functions displayed, you can see those from CProduce as well as those in CFruit. Experiment with the filters to see how they affect the display.

FIGURE 6.38

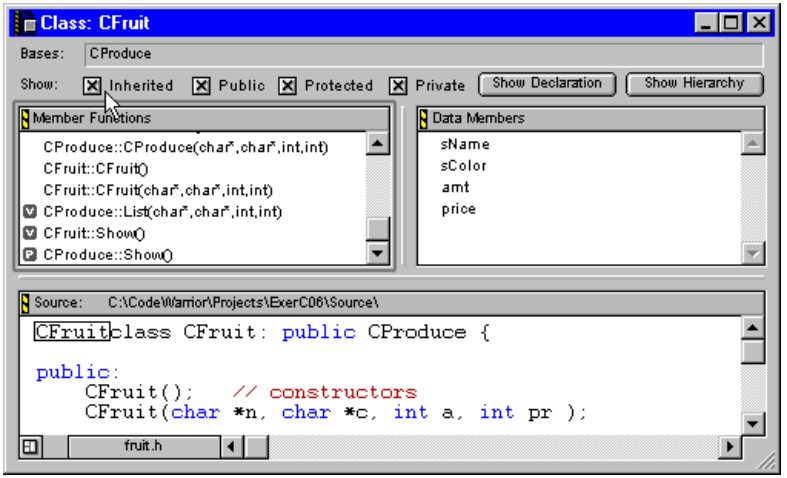
Open a single-class browser for the CFruit class.




Also note that this view has some special navigational features. Click the Show Declaration button to view the class declaration in the Source pane. Also click the Show Hierarchy button to see a single-class hierarchy for the CFruit class. You can close this view when you're through with it if you wish.

FIGURE 6.39

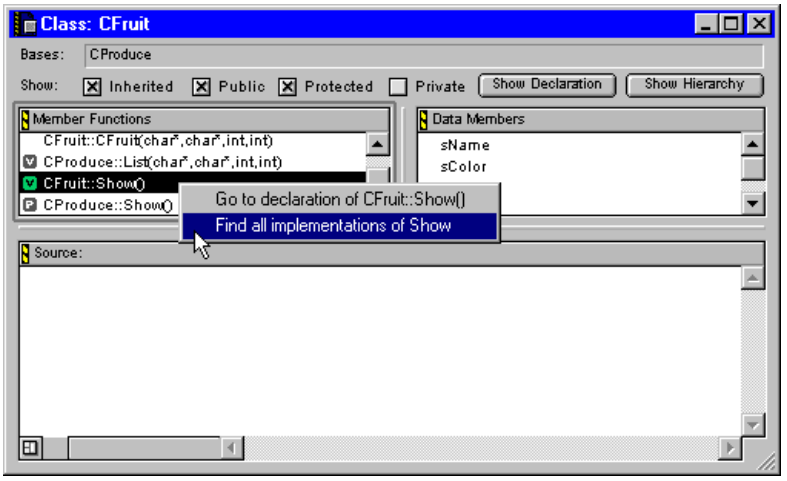
The single-class browser lets you filter what you see.



Finally, display the context menu for the **Show()** function, and use that path to show all implementations of this function. Figure 6.40 shows one way to do this.

FIGURE 6.40

Use the context menu to open a symbol window for the Show() function.

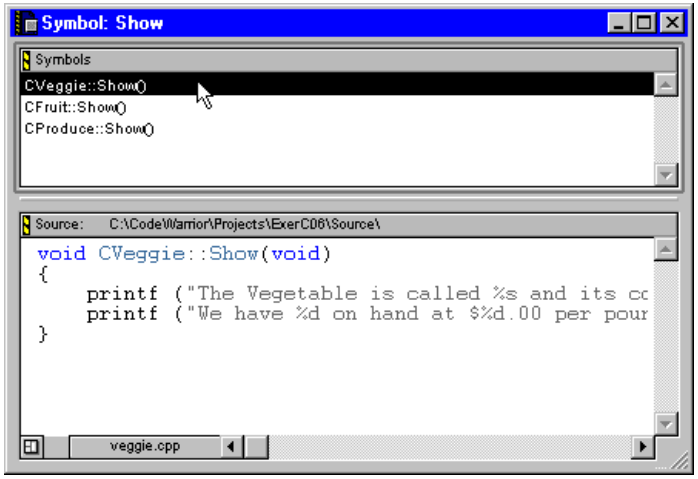



**8. Use the symbol window.** The symbol window lists all occurrences of a particular symbol in the catalog. Figure 6.41 shows you the list of three **Show()** functions that occur in this catalog—one each for CProduce, CFruit, and CVeggie. Select one or another of these items, and the related source code appears in the Source pane.

If you select the **CProduce::Show()** function, nothing appears in the Source pane because this is a pure virtual function—in C++ syntax, that's a function that has no definition. If you want, use the browser to locate the declaration of **CProduce::Show()**. You'll see that the prototype for this function says it is equal to zero—the C++ syntax for declaring a pure virtual function. Subclasses must override such a function, and both CFruit and CVeggie do, and you can see those implementations right here in the symbol window.

FIGURE 6.41

The symbol window lets you pick which instance of a symbol you're interested in looking at.



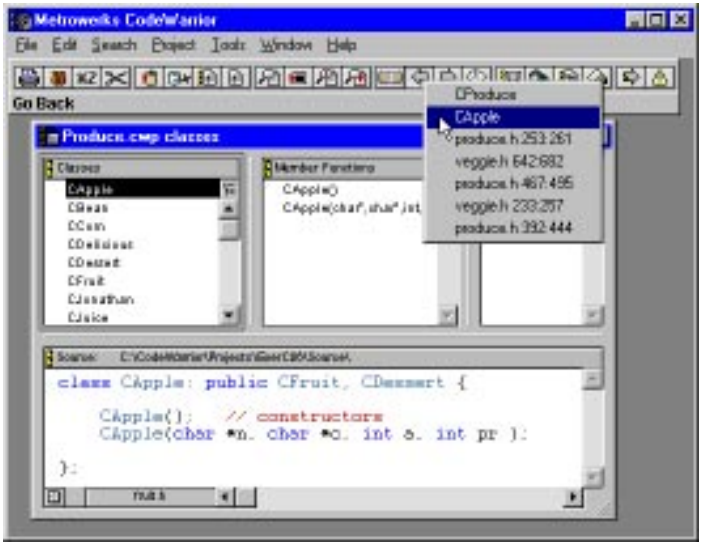

Once again, the information on display in the Source pane is limited. To see the complete source file in an editor window, double-click an item in the list, or use the file button at the bottom of the window. Try both techniques!

**9. Travel through browser views chronologically.** For a final treat, in this step you use the Go Back and Go Forward commands in the Search menu and the toolbar to see other browser views you have already visited.

Simply choose Go Back (Shift+Ctrl+B) a few times from the Search menu and watch what happens. You walk backwards through every view you have had on display as a result of browser actions—including editor views. Once you have walked back a few steps, you can Go Forward (Shift+Ctrl+F) as well.

Now, click and hold on either the Go Back or the Go Forward buttons in the toolbar to see a complete list of available views, as shown in Figure 6.42. Your list is likely to vary from this one, but you can select any item from the list to jump right to that view.

FIGURE 6.42
You can see a list of previous browser views from the toolbar.




When you're through, you can close all windows and exit from CodeWarrior if you wish. You have completed this exercise and you have sailed through the browser from one end to the other. Because this is your first look at the browser, it might still seem more like a maze than a map. That's because the browser gives you multiple possible routes to a single goal, and in the beginning the plethora of paths might seem confusing. As you settle in and use the tool regularly, you'll carve your own favorite paths. In the end, the way you use the browser will be unique to your own personal style of coding and work, because the browser is flexible enough to adapt to you, rather than forcing you to adapt to it. And *that* is very cool.

## Summary

The CodeWarrior browser is a fantastic tool. This chapter introduced you to every aspect of the browser. You met the browser interface, and learned how to use it to move through your code at will. You saw how

to use the browser to accomplish a variety of common tasks, such as examining a class hierarchy, studying various implementations of an overridden function, navigating code using the context menu, and more.

Recall that the browser has a series of viewing styles—catalog, hierarchy, and class browser—that you choose in order to look at your code from either a broad or narrow perspective. You can switch among these views quickly and easily.

Like all great ideas, the browser is simple in concept and implementation. Yet there are virtually no limits to what you can do with it. The real power of tools like the browser lies in its flexibility, and in your imagination. You are quite likely to have a unique set of tasks to accomplish. Because the browser is so flexible, you should have no trouble using it to meet your goals.

With this chapter we have completed our discussion of the tools you use to write code in the IDE—the project manager, the editor, and the browser. Next on the tour is how to compile and link your code. After that, we'll examine the CodeWarrior debugger in depth.

# INDEX

**Note to the Reader:** Throughout this index page numbers in **bold** indicate primary discussions of a topic. Page numbers in *italic* indicate illustrations.

## Symbols


& (ampersands) in regular expressions, **198–199**
> (angle brackets) for included files, 83, 297
(asterisks) in regular expressions, **194–195**
\ (backslashes) in regular expressions, **197–198**
{} (braces)
balancing, **129–131**, 150–151, *151*
for folders, 82
[] (brackets)
balancing, **129–131**
in regular expressions, **196**
^ (carets) in regular expressions, **196–197**
- (dashes)
for breakpoints, 359, 398
in regular expressions, **196**
in Source Browser window, 342–343
$ (dollar signs) in regular expressions, **197–198**
# (number signs) for preprocessor statements, 299
() parentheses
balancing, **129–131**
for folders, 83, 87, 297
in regular expressions, **195–196**
. (periods) in regular expressions, **194**
+ (plus signs) in regular expressions, **195**
" (quotation marks) for included files, 83, 297
; (semicolons)
compiler warnings for, 309
errors involving, 316–317


## A


A6 register, 564
absolute access paths, 296
Abstract Windowing Toolkit (AWT), 17, 440
academic pricing, 595
accelerators, 4
acceptSuspendResumeEvents flag, 572
Access Paths panel, **81–84**, *82*, 178, *178*, 293–296, *295*
Acrobat Reader, 29
Activate Browser option, 224, 262–263, *263*
Add Files command, 64, 94, 321, 417, 429, 457–458
Add Files dialog box, 64, *64*, 94
Add marker command, 132, 154, *154*
Add Window command, 63, 93
algorithms, C++ library for, 421
aligning
bytes, 517
layout panes, **483–485**, *483–484*
structures, 562, 567
Alignment Palette command, 483
Alt key for copying text, 147
ampersands (&) in regular expressions, **198–199**
angle brackets (<>) for included files, 83, 297
ANSI C standard, 291, 414
Ansicppx86.lib file, 421–422
AnsiCx86.cwp file, 89
Ansicx86.lib file, 94, 417, 429
Apfcvt.exe utility, 601
APIMon.exe utility, 601

APPL file type, 576
AppletFrame.java file, 545, 550
applets, **543–544**, *544*
  classes for, 447
  classes.zip file for, **546–547**, *546*
  project settings for, 544
  running, **547**, *548*
  source files for, **544–545**, *545*
application frameworks, **17**, **440**
  exercises for, **456–460**
  fundamentals of, **17–19**, **441–442**
  Java, **447–451**, *448–449*, *451*
  MFC, **443–446**, *443*, *446*
  PowerPlant, **451–456**, *454*
Application Programming Interface (API), 12
  for Java Virtual Machine, 447
  plugin, 280
applications
  building, **530–531**, **574–575**
  file type for, **576**
  Java programming for, **548–552**, *549–550*
  Mac OS programming for, **570–578**, *571*, *573–574*, *577*
  moving to MAC OS computers, **575–576**
  paths for, **551–552**
  project settings for, **528–529**, 549, **571–572**
  running, **530–531**, **552**, **577**, *577*
  source files for, **529–530**, *530*, **550**, *550*, **572–574**, *573*
  Windows programming for, **528–531**, *529–530*
APPMODUL.CPP file, 531
Arrange Objects command, 484
Arrange Objects dialog box, 484–486, *484*
arranging
  layout panes, **481–486**, *482–484*
  windows, **137–138**, 155–156
Array window, 377, *377*, 381
arrays
  frameworks for, 18
  in memory, 381, *381*
  Pascal library for, 424
  variables for, **376–377**, *376–377*, 402–403, *403*
arrow keys
  for insertion point, 114–115
  in selecting text, 117
arrow pointers in searches, 180, *181*
assembly language, viewing, **382–383**, *383*
*Assembly Language Reference*, 523
assignments, compiler warnings for, 308
asterisks (*) in regular expressions, **194–195**
Auto Indent option, 123
automatic file saving, **139–140**
Automatically Launch Applications When SYM File Opened option, 349

# B

back-end compiler tasks, 21
background color
  for pixels, 495
  for text, 126, 150
backslashes (\) in regular expressions, **197–198**
backward searches, 165, 172–174, 201
Balance While Typing option, 130
balancing punctuation, **129–131**, 150–151, *151*
Base Address option, 516
base classes
  in multi-class window, 228–229
  in single-class window, 239
Batch option, 186
batch reports, **185–186**, 207–210, *208–210*
  activating, **186–187**, *186*
  Search Results window with, **187–191**, *189*, 208–209, *209*
beeping
  in balancing punctuation, 130
  in searches, 172
bin directory, **600–601**
binary files, deleting, 100, 288
binhex conversion program, 578
BinSDK directory, **601–602**
bit manipulation, Pascal library for, 424
bitmaps, **493–495**, *493*
  color and patterns in, **497–498**
  editing pixels in, **497**
  sample views for, **496–497**

tools for, **495–496**
transparency in, **499–500**, *499*
black triangles in PowerPlant, 454
bold print in Project window, 55
books, CodeWarrior, 603
braces ({})
balancing, **129–131**, 150–151, *151*
for folders, 82
brackets ([])
balancing, **129–131**
in regular expressions, **196**
brackets (<>) for included files, 83, 297
Break on C++ Exceptions command, 366–367
Break on Java Exceptions command, 366–367
breakpoints
conditional, 360, **365–366**, 399–401, *400–401*
for exceptions, **366–367**
fundamentals of, **360**
for libraries, 350, 432, *433*
as markers, 365
missing, **362–363**
in Program window, 345
setting and clearing, **359–361**, *360*, 394, *395*, 398–401, *399–401*
in Source Browser window, 342–343, 357
temporary, **360–361**, 394, *395*
viewing and controlling, **364–365**
Breakpoints window, 364–365, *364*, 399–400, *400*
Bring Up To Date command, 98, 155, 262–263, 288, 315
browser, **216–224**
activating, **224–225**, *225*, 262–263, *263*
background of, **221–222**
catalog view in, **222–228**, *226*, 263–265, *264–265*
class declarations in, **255–256**
deep and wide disclosures in, **233–234**, *234*
exercises for, **262–274**
functions in, **256–257**
graphical view in, **228–239**, *229–232*, *234–237*, *239*
for Java, 451, *451*
for MFC, 445, *446*
for multi-class browsing, **239–249**, *240*, *242–244*, *246–248*, 268–270, *268–270*
for multi-class hierarchies, **228–238**, *229–230*, *234–237*, 265–266, *266*
multiple inheritance in, **234–235**, *235–236*
navigating code in, **254–257**
reviewing history in, **258–259**, *259*, 273–274, *274*
settings for, **260–262**, *260–261*
showing and hiding classes in, **230–232**, *231–232*
for single-class browsing, **249–252**, *251*, 270–272, *271–272*
for single-class hierarchies, 236, *237*, **238–239**, *239*, 266–268, *267*
Symbol window in, **252–254**, *253*, 272–273, *273*
views in, **222–224**, *223*, **254–255**
Browser Coloring panel, 260, *260*
Build Extras panel, **79–80**, *81*, 224–225, *225*
Byte Alignment option, 517
byte manipulation, Pascal library for, 424
bytecodes
.class files for, 542, **546–547**
in disassembling, 298
from Java compiler, 22, 538–539


# C


C, libraries for, 16, **414–419**, *417*
C++, libraries for, 16, **419–423**, *421*
C/C++ Compiler panel, 92, 290–292, *291*
C/C++ Warnings panel, 308–309, *309*
.c extension, 286
.c++ extension, 286
C String format command, 373, 402
C strings for variable values, 372
calling chains
examining, 350
fundamentals of, **344**
walking through, **355–356**, *356*, 395, *396*
canBackground flag, 572
Capscrn.exe utility, 601
capture video utility, 601
carets (^) in regular expressions, **196–197**
case-sensitivity in searches, 170, 357
.cat extension, 225

catalog view, 218–219, **222–228**, *226*, 263–265, *264–265*
for class declarations, 255
for functions, 256
.cc extension, 286
CDessert class, 262
central classes, 238–239, *239*
CFM68K Code Fragment Manager, 560
ChangeMe.c file, 89, 93, 99–100
ChangeMe.h file, 89, 99–100
changes, undoing, **120–122**, 146
character constants for variable values, 372
Character format command, 373
characters
C library functions for, 416
in regular expressions, **194**, 196–197
for variable display, 373
check marks
in editor, 139, *139*, 144, *145*
in Project window, 55–58, 70–71, 286, *287*
Check Syntax command, 289
choosing vs. selecting, 4
circles
for breakpoints, 359, 361
drawing, 496
class browsing, **10–11**, *11*, 219, 222–224, **239–242**, *240*, *242*, 268–270, *268–270*
for class declarations, 256
Classes pane in, **242–244**, *243–244*
Data Members pane in, **246**, *247*
for functions, 257
Functions pane in, **244–246**, *246*
Source pane in, **241–242**, *242*, **247–248**, *248*
for viewing and editing code, **248–249**
class declarations in browser, **255–256**
.class extension, 286, 542
classes
browsing. *See* class browsing; single-class browsing
deep and wide disclosures in, **233–234**, *234*
hierarchy of. *See* multi-class hierarchy view; single-class hierarchy view
importing, **450–451**, *451*
multiple inheritance in, **234–235**, *235–236*
in object-oriented code, 217–218
showing and hiding, **230–232**, *231–232*
Classes pane, **242–244**, *243–244*
classes.zip file, **448–450**, **546–547**, *546*
Clear command
for layout panes, 480
for menus, 489
Clear All Breakpoints command, 359
Clear Current Watchpoint command, 386
Clear Toolbar command, 39
Clear Watchpoint command, 386
clearing
breakpoints, 359, 364
keyword lists, 127
layout panes, 480
menus, 489
toolbar, 35, 39
watchpoints, 386–387
Close command, 60, 142
Close All command, 142
closing
files in editor, **142**
groups, 65
projects, **60–61**, 100
CObject class, 452
code
in browser, **248–249**, *250*, **254–257**
compiling and linking. *See* compilers and compiling; linkers and linking
converting between platforms, 134
debugging. *See* debugger and debugging
disassembling, **297–299**, *298*
finding, **357–358**, *357*
navigating through, **355–358**, *356–357*
object-oriented. *See* object-oriented code
reusing. *See* libraries
running, **348–350**, *349*
skipping and repeating, **353–355**, *354*
stepping through, **347–355**, *347*, *353–354*
viewing and editing, **248–249**, *250*
Code column, 55, 288

Code Fragment Managers, 560
Code Model option, 562
code resources, 560
CodeView format, 329
  setting for, 334
  with Visual C++, 337
  for Windows programming, 518
CodeView Format option, 518
CodeWarrior Books directory, 603
*CodeWarrior C/C++/Asm Language Reference*, 292, 309
CodeWarrior Documentation directory, **602**
*CodeWarrior Error Reference*, 307
CodeWarrior Examples directory, 603
CodeWarrior Lite demo version, 88, 594
CodeWarrior Lite Setup application, **28–31**, *29–31*
*CodeWarrior Pascal Language Reference*, 292, 309
*CodeWarrior QuickStart* manual, 594
CodeWarrior Reference Web Site, 600
COFF object format, 522
Collapse All command, 369
color
  for bitmaps, **497–498**
  for browser, 260, *260*
  for pixels, 495–496
  for text, **126–129**, *127–129*, 148, 150, 500
color swatch, 126
color tool, 495
coloring
  register, **370–371**
  syntax, **126–129**, *127–129*, 148
Colors menu, 498
Command File option, 522
command IDs for menu items, 487, 490, *491*, 507
comments, color for, 126
Committed Heap Size option, 516
Committed Stack Size option, 516
Common SubExpressions option, 520
comparisons, compiler warnings for, 309
compatibility for Windows programming, 523
Compile command, 288, 314
Compiler column, 283
Compiler pop-up menu, 283
compiler relative access paths, 296
compilers and compiling
  architecture of, **21–24**, *21*
  check marks for, 55–56, 70–71, 286, *287*
  choosing, **278–286**, *279*, *281*, *283*
  code generation control in, **289–290**
  commands for, **288–289**, 314–315
  disassembling code, **297–299**, *298*
  errors and warnings with, **307–314**, *309–311*
  exercises for, **313–323**
  extensions for, 78, 283–286
  fundamentals of, **47**
  headers, **300–303**, *301–302*
  just-in-time, 331, 538–539
  language, **282–286**, *283*, **290–293**, *291*, *293*
  plugin, **279–280**, *279*
  preprocessors for, **299–300**, *300*
  resource, **522**
  selecting files for, **286–287**, *287*
  speeding up, **293–297**, *294–296*
  target, **280–282**, *281*
Condition column, 366
conditional breakpoints, 360, **365–366**, 399–401, *400–401*
Confirm "Kill Process" When Closing or Quitting option, 351
Console User Interface (CUI), 92, *92*
constants
  in browser, 218, 226–227
  for variable values, 372
Constructor, **464**
  background of, **464–465**
  for bitmaps, **493–500**, *493*, *499*
  exercises for, **502–508**
  help files for, 602
  launching, 502–503
  for layouts, **476–486**, *478–480*, *482–484*
  for menus, **486–492**, *487–489*, *491*, 506–508, *507*
  operation of, **465–467**, *466*
  resource management in, **469–476**, *470–471*, *474*
  for text traits, **500–501**, *501*, 503–504, *504*
  working with, **467–468**, *468*
Constructor.exe file, 601
*Constructor Manual*, 466, 480

constructors, debugging, 349
containers, C++ library for, 420
containment in layouts, **477–479**, *478–479*
context menus
    in class hierarchy view, 237–238, *237*, 266–267, *267*
    in Classes pane, 269, *269–270*
    in Functions pane, 245–246, *246*
    in Members pane, 246
    settings for, 261
    in Source pane, 244, 248
    for Symbol window, 252, 272, *272*
Context Popup delay option, 261
Control menu commands
    Break on C++ Exceptions, 366–367
    Break on Java Exceptions, 366–367
    Clear All Breakpoints, 359
    Kill, 348, 351
    Run, 343, 348, 350
    Step Into, 348, 352–353, 394
    Step Out, 348, 353
    Step Over, 348, 352, 393
    Stop, 348, 350–351
converting code between platforms, 134
Copy/Const Propagation option, 520
Copy to Expression command, 375, 403
copying
    in Constructor, 468
    pixels, 497
    project files, 60
    resources, 473
    text, 147–148, *147*
Courier New font, 123
.cp extension, 286, 526
.cpp extension, 286
CProduce class, 262
crashes from user settings
    of current statement arrow, 354
    of memory attributes, 474
    of register values, 382
    of variable values, 369
Create Folder option, 53–54, 88
Create New Resource dialog box, 469–471, *470–471*, 504, *504*
Creator option, 559
cross-platform programming, **585–586**
    debugger for, **330–332**
    stationery for, 51–52
Ctrl key
    in file selection, 62
    for insertion point, 114–115
    in text selection, 117–118
ctype.h file, 416
curly braces ({})
    balancing, **129–131**, 150–151, *151*
    for folders, 82
current statement arrow, 345, **353–355**, *354*, 393, *394*
Custom Keywords panel, 128–129, *128*
Cut command
    for layout panes, 480
    for menus, 489
    for resources, 473
    for text, 120
CW Release Notes folder, 603
.cwp extension, 53–54
%CWPROJPATH% symbol, 551
cycling through windows, 138, 155

# D

dashes (-)
    for breakpoints, 359, 398
    in regular expressions, **196**
    in Source Browser window, 342–343
Data column, 55, 288
data forks, 91
data hiding, **219–220**
Data Members pane
    in multi-class browser, **246**, *247*
    in single-class browser, **250–251**, *251*
Data menu commands
    Clear Current Watchpoint, 386
    Clear Watchpoint, 386
    Collapse All, 369

Copy to Expression, 375, 403
for data types, 373
Open Array Window, 377
Open Variable Window, 374
Set Watchpoint, 385
Show Types, 372, 402
View Memory, 380, 385
View Memory As, 381
data size options, 562
data types, **372–374**, *373*, 401–403, *402–403*
.dbg extension, 378–379
Dead Code Elimination option, 520
dead code stripping
for Mac OS programming, 564, 568
missing breakpoints from, 362–363
for Windows programming, 520
Dead Store Elimination option, 520
Dead Strip Static Initialization Code option, 564
Debug column, *55*, 57, *72–73*, *72*, 333, *334*
Debug command, 288, 304–305, 336–337, 432
debugger and debugging
activating, **329**
for assembly language, **382–383**, *383*
background of, **330–332**
breakpoints in, **359–367**, *360*, *364*, 394, *395*, 398–401, *399–401*
exercises for, **389–405**
help files for, 602
killing programs in, **351**
launching, **336–337**, *338*, 391, *392*
libraries, 350, **384**, 432–435, *433–435*
log window in, **346**, *346*, 391
low-level features in, **380–383**, *381–383*
Mac OS programming, **563–564**, **578–579**
for memory, **380–381**, *381*
for multiple languages, **387**
for multiple projects, **388–389**
for multitasking, **387–388**, *388*
navigating through code in, **355–358**, *356–357*
operation of, **328–329**
Program window in, **343–345**, *343*, 391, *392*
for registers, **382**, *382*
running code in, **348–350**, *349*
saving debug information for, **378–379**
setting up, **332–336**, *333–335*, 390
skipping and repeating code in, **353–355**, *354*
Source Browser window in, **340–343**, *340*
stepping into routines in, **352–353**, 394–395, *396*
stepping out of routines in, **353**
stepping over routines in, **352**, 393
stepping through code in, **347–348**, *347*, 353, *353*
stopping program in, **350–351**
switching with source code, **358–359**
symbol information for, **71–73**, *72*
viewing and changing data in, **367–377**, *368*, *373–375*, *377–378*, 397–398, *398*, 403–404, *404*
watchpoints in, **384–387**, *386–387*
windows for, **339–340**
Windows programming, **531–534**, *532–533*, *535*
Debugger Plugins directory, 601
decimal digits for variable values, 371
declarations
in browser, **255–256**
defined, 5
vs. definitions, 227
in Source pane, 244
deep disclosures in browser, **233–234**, *234*
Default format command, 373
default options
for projects, **75–76**
stationery for, 52
Default Size for Unbounded Arrays option, 377
#define statements, 299
definitions
vs. declarations, 227
defined, 5
deleting
binary files, 100, 288
breakpoints, 359, 364
compiler list extensions, 284
Expression window variables, 376
file sets, **184–185**, *184*
files and groups, **66–67**
files from searches, **180**

keywords, 127
layout panes, 480
markers, 133
menu items, 491
menus, 489
pixels, 497
resources, 473
with search and replace, 161
text, **120**
watchpoints, 386–387
dependency lists, 80
diagnostics, C++ library for, 420
diagonal lines in browser window, 229, *230*
dialog boxes, utility for, 601
direction in search and replace, 162, 165, 201
disabled commands, 34–35
Disassemble command, 298
disassembling code, **297–299**, *298*
disclosure triangles
for classes, 230–233, *231*, *234*, 265–266, *266*
for Constructor, 467, 477
in Errors & Warnings window, 312
for group names, 55, 65
for layouts, 477
for multi-file searches, 176
in Program window, 345, 347, 368, 402
in single-class hierarchy view, 239
for stationery, 51
for structures, 368–369
Display Classes window, 471–472, *471*, 502
distributing layout panes, **485–486**
DlgEdit.exe utility, 601
DLLs. *See* dynamically linked libraries (DLLs)
documentation
for compilers, 292
directory for, **602**
for libraries, 415
for PowerPlant, 456
for projects, 48
Documentation Overview file, 602
doesActivateOnFGSwitch flag, 572
dollar signs ($) in regular expressions, **197–198**
Dow, Gregory, 452–453
drag-and-drop
for bitmaps, 497
in Constructor, 467–468, 473, 479
for groups, 65
for menus, 489
for moving text, 118–119, *118*, 147–148, *147*
in panes, 125
Drag and Drop Editing option, 119
drawing mode for text, 500
dropper tool, 494, 496
dummy files, 85
Duplicate Resource command, 473
dynamically linked libraries (DLLs), **57**
building, **527–528**
debugging, **384**
exporting functions for, **526–527**, *527*
project settings for, **524–525**
source files for, **525–526**, *526*
Windows programming for, **523–528**, *524*


# E


Eclipse.cp file, 525–526
Eclipse.cwp file, 428–436, *430–435*, 528
Eclipse.dll file, 527, 533
Eclipse.dll.lib file, 527, 529
Eclipse.rc file, 529
EclipseApp.cp file, 529–530, 532
EclipseDLL.cwp file, 524, *524*
Edit menu commands, 33
Clear, 480, 489
Cut, 120, 473, 480, 489
Duplicate Resource, 473
Edit Resource, 473
Find, 357
Grid, 483
Menu Info, 489
Menu Item Info, 491
for moving text, 118
New Menu, 488, 507
New Menu Item, 490, 507

New Resource, 469, *470*, 503
New Separator Item, 490
Preferences, 38, 74, 119
Project Settings, 76, 89, 127
Redo, 120–121, 146
Resource, 473
Resource Info, 474
Shift Left, 123, 152
Shift Right, 123, 152
Undo, 120–121, 146
editing
bitmaps, **493–500**, *493*, *499*
code, **248–249**, *250*
in Constructor, 468, *468*
keyword lists, **126–129**, *128*
layouts, **476–486**, *478–480*, *482–484*
libraries, 427
memory attributes for resources, 474
in Memory window, 380
menus, **486–492**, *487–489*, *491*, 506–508, *507*
object properties, 505–506, *506*
pixels, **497**
in Search Results window, **190–191**
in Source pane, 247
text traits, **500–501**, *501*, 503–504, *504*
editor, **111**
adding text with, **113–114**, 144–145
appearance of, **135–136**, *135*
background of, **111–112**
closing files in, **142**
deleting text in, **120**
Editor window in, **112–113**, *113*
exercises for, **144–157**
fonts in, **122–123**, 148
indenting in, **123–124**, 151–152, *151*
jumping to routines in, **131–132**, *131*
line endings in, **134**, *134*
markers in, **132–134**, *133*, 154, *154*
moving insertion point in, **114–116**
moving text in, **118–119**, *118–119*, 147–148, *147*
multiple windows in, **137–138**
printing files in, **143–144**
punctuation in, **129–131**, 150–151, *151*
related files in, **136–137**, *136*
saving files in, **138–142**, *139*, 145–146
search and replace feature in. *See* search and replace feature
selecting text in, **117–118**, 145
splitting panes in, **124–125**, *125*, 152–153, *152*
syntax coloring in, **126–129**, *127–129*, 148
undoing changes in, **120–122**, 146
Editor Settings panel
for automatic saves, 140
for balancing punctuation, 130
for content menus, 261, *261*
for drag-and-drop, 119, *119*, 149
for multiple undo/redo, 122
for routine sorting order, 132
Editor window, 109, *110*, **112–113**, *113*, 298, *298*
empty oval tool, 495
empty rectangle tool, 494
empty rounded rectangle tool, 494
Enable Debugger command, 332–333, *333*, 336, 390
encapsulation, 217–218, 220
End key in editor, 115
Enter 'Find' String command, 166
Enter key in editor, 114
Enter 'Replace" String command, 167, 173
Entire Word option
for regular expressions, 193
for search and replace feature, 170–171, 202, *203*
Entry Point Usage pop-up menu, 520, 525
Entry Points section, 569–570
enumerated constants, 218, 226–227
Enumeration format command, 373
EOLN Format command, 134
eraser tool, 494–495, 498
errors
breaking on, **366–367**
compiler, **307–314**, *309–311*
handling, **312–313**
linker, 304
messages for, 12, *13*, **310–312**, *311*, 315–319, *316–318*
Errors & Warnings window, 12, *13*, **310–312**, *311*, 315–319, *316–318*

events
frameworks for, 441–442
Log window for, **346**, *346*, 391
exceptions, breaking on, **366–367**
Exer03 folder, 89
Exer04 folder, 144
Exer05 folder, 199
Exer06 folder, 262
Exer07 folder, 314
Exer08 folder, 390
Exer09 folder, 428
Exer10 folder, 457
Exer11 folder, 503
Exer12 folder, 528
Exer13 folder, 543, 549
Exer14 folder, 571
exercises
for browser, **262–274**
for compiling, **313–323**
for Constructor, **502–508**
for debugger, **389–405**
for editor, **144–157**
for frameworks, **456–460**
for menus and toolbar, **36–40**
for projects, **87–101**
for search and replace feature, **199–212**
for standard libraries, **428–436**
exporting functions for DLLs, **526–527**, *527*
Expression window, **375–376**, *375*, 403–404, *404*
extensions for compilers, 78, 283–286

# F

Far options, 562
Fast Link option, 564
fat applications, 556, 588
file changed icon
in Editor window, *113*
in Errors & Warnings window, 312
File menu commands, 33, 49, *50*
Close, 60, 142
Close All, 142
New, 109
New Project, 50, 88, 96
New Project File, 469
Open, 58, 68
Open File, 110
Open Selection, 137
Print, 144
Print Setup, 143
Revert, 141
Save, 139, 145
Save a Copy As, 60, 141, 191, 313
Save All, 139
Save As, 140–141
Save Settings, 378
File name extension column, 282
File Name option
for Mac OS programming, 559
for Windows programming, 515
file names in Project window, 55, *55*
file sets, **182**
choosing, **184**
creating, **182–183**, *183*
deleting, **184–185**, *184*
saving, 182, 206–207, *207*
File Sets menu commands, 176
Remove a File Set, 184
Save this File Set, 182, 206
File type column, 282
file type for applications, **576**
files
adding to projects, **62–64**, *64*, 93–95, *93–95*
binary, 100, 288
closing, **142**
for compiling
searches for, 295–297, *295*
selecting, **286–287**, *287*
creating, **109**, *110*
deleting, **66–67**
header. *See* header files
listing, 68, *68*

moving, **66**, *66*
opening, **67–68**, *68*, 99, **110–111**, 136–137, *137*, 149–150, 247, *248*
prefix, 292–293, 301, 321–323, *322*
printing, **143–144**
related, **136–137**, *136*
renaming, **140–141**
reverting, **141**
saving, **138–142**, *139*, 145–146
searching in, **175–185**, *177–179*, *181*, *183–184*
selecting
for compiling, **286–287**, *287*
for debugger, 333, *334*
for projects, **61–62**, *62*
touching, **70–71**, *71*
tracking. *See* projects and project manager
windows for, **108–109**
filled oval tool, 495
filled rectangle tool, 494
filled rounded rectangle tool, 494
filters in single-class browser, 250–252, 270, *271*
final methods, 542
Find command, 162, 171, 186–187, 201, 357
Find all Implementations of command, 255
Find dialog box, 357–358, *357*
Find in Next File command, 182
Find Next command, 165, 171, 200
Find Previous command, 165, 171, 201
Find Previous Selection command, 165
Find Selection command, 164–165, 200
Find window, 162–164, *163*
for batch reports, 186, *186*, 207–209, *208*
for file sets, 206–207, *207*
for finding and replacing text, **171–174**
for multi-file searches, 176–177, *177*, 205–206, *206*
opening, 201, *201*
for range of searches, **169**
for regular expressions, 192–193, *192*, 211–212, *211*
for replacing found items, 204–205, *205*
for scope of searches, **170–171**
for search and replace strings, **166–168**, *167*, 202, *202*
for whole words, 202, *203*
for wrapping in searches, 203–204
finding
code, **357–358**, *357*
editor text. *See* search and replace feature
files for compilation, 295–297, *295*
Fixed format command, 373
Flashing Delay field, 130
floating-point digits for variable values, 372
Floating Point format command, 373
floating-point math routines
C library for, 416
Pascal library for, 424
floating-point unit (FPU) registers, 382
flow control, frameworks for, 442
focus
in class browser, 241
in Search Results window, 191
focus bar for moving files, 66
folders
braces for, 82
in CodeWarrior Gold, 600–605
for installation, 29
parentheses for, 83, 87, 297
for projects, 53
for shortcuts, 30
Font & Tabs panel, 122–123, *122*, 148, *148*
FontEdit.exe utility, 601
fonts
in editor, **122–123**, 148
for menu items, 492
saving settings for, 141
for text, 500, 503–504, *504*
utility for, 601
foreground color for pixels, 495–496
Fract format command, 373
frameworks, **17**, **440**
exercises for, **456–460**
fundamentals of, **17–19**, **441–442**
Java, **447–451**, *448–449*, *451*
MFC, **443–446**, *443*, *446*
PowerPlant, **451–456**, *454*
front-end compilers
settings for, **290–293**, *291*, *293*
tasks for, 21

Full Path in SYM Files option, 564
function pop-up menu, 312
function prototypes
in class browser, 248–249, *250*
as declarations, 227
functions. *See* routines
Functions.p file, 428
Functions pane
in multi-class browser, **244–246**, *246*
in single-class browser, **250–251**, *251*

# G

Gdi32.lib file, 94
general utilities
C library for, 415
C++ library for, 420
Generate A6 Stack Frames option, 564
Generate CodeView Info option, 334, 521, 525, 529
Generate Link Map option
for Mac OS programming, 564
for Windows programming, 521
Generate Make Map File option, 80
Generate SYM File option
for Mac OS programming, 564
for Windows programming, 306, 334, 521
Global Optimization option, 568
global variables
in browser, 218, 226–227
in debugger, 357, **367–368**, *368*, 397–398, *398*
windows for, 374–375
Globals pane, 367–368, *368*, 397
Go Back command, 258, 273
Go Forward command, 258, 273
Go-To-Line button
in Editor window, *113*
in Errors & Warnings window, 312
Go To Line Number dialog box, 115, *115*
graphical browser
deep and wide disclosures in, **233–234**, *234*
for multi-class hierarchies, **228–238**, *229–230*, *234–237*
multiple inheritance in, **234–235**, *235–236*
showing and hiding classes in, **230–232**, *231–232*
for single-class hierarchies, 236, *237*, **238–239**, *239*, 266–268, *267*
graphical user interfaces (GUIs)
C and Pascal support for, 414
Constructor as builder of, **464–465**
frameworks for, **441–442**
Java classes for, 447
grep, 192
grids for layout panes, **481–483**, *482–483*
Group names in Project window, 55, *55*
groups
in Constructor, 467
creating and naming, **64–65**, *65*, 95, *96*
deleting, **66–67**
listing files in, 68, *68*
moving, **66**, *66*
searching in, **175–185**, *177–179*, *181*, *183–184*

# H

.h extension, 286
hard drives and compilation speed, 297
header files, 5
including, **69–70**, *69*, 99–100, *99*, 418, 430–431, *431*
in MFC, **444–445**
in PowerPlant, **454**, *454*
precompiled, 56, 284, **300–303**, *301–302*, 319–321, *321*
prefix files for, 292–293, 301, 321–323, *322*
searching in, **177–179**, 206
Headers directory
for Mac OS, 604
for Windows, 605
heap
in Mac OS programming, 559
monitoring, 601
in Windows programming, 516
Hello.c file, 573

Hello.cwp file, 571
Hello.rsrc file, 573
(Help) directory, 602
help for toolbar icons, 34, 38
Help menu, 33
hexadecimal digits for variable values, 371
Hexadecimal format command, 373
Hide Toolbar command, 38
hiding
  classes, **230–232**, *231–232*
  data, **219–220**
  errors and warnings, 311
  grids, 481–482
  icons, 135
  toolbar, 35, 38
hierarchical variables, **368–369**
hierarchies
  of classes. *See* multi-class hierarchy view; single-class hierarchy view
  in layouts, **477–479**, *478–479*
highlighting, 4
hits in searches, 164, 189–191
home.htm file, **600**
Home key in editor, 115
horizontal orientation in class browser, 242
hosts, 5
hot spot tool, 495
.html extension, 544
human interface model, **116**
Hw32.exe utility, 601
hyphens (-)
  for breakpoints, 359, 398
  in regular expressions, **196**
  in Source Browser window, 342–343

# I

icons in editor, **135**, *135*
Ide.exe file, 601
IDs
  for menu items, 487, 490, *491*, 507
  for menus, 489–490
  for resources, 472–473
  for submenus, 492
Ignore Case option, 170–171
Ignored By Make flag, 283–285
implementations, 5
Import "Weak" command, 449–450, 546
importing Java classes, **450–451**, *451*
In Variable Panes, Show Variable Types By Default option, 372
#include statements and including
  for C libraries, 418, 430–431, *431*
  compilation options for, 297
  header files, **69–70**, *69*, 99–100, *99*
  for MFC, **444–445**
  for PowerPlant, 454
  precompiled files, 56, 284, **300–303**, *301–302*, 319–321, *321*
  syntax choices for, 83
indenting in editor, **122–124**, 151–152, *151*
infinite loops, 351
Info dialog box, 474, *474*
inheritance
  browser display for, **234–235**, *235–236*
  in object-oriented code, 218
  in PowerPlant, **452–455**, *454*
inherited functions, viewing, 251–252, 270, *271*
initialization code
  in Mac OS 68K programming, 559
  in Mac OS PowerPC programming, 569
Initialize Before command, 449–450, 547
inline assembly support, 292, 523
input/output routines
  C library for, 415
  C++ library for, 421
  Java classes for, 447
  Pascal library for, 424
insertion point, 113
  fundamentals of, **114**
  human interface model for, **116**
  moving, **114–116**
  saving position of, 142
installation, **28–31**, *29–31*
installer, 28, *29*
instances, 217–218
Instruction Scheduling option, 568

instruction sets for virtual machines, 23, **538–539**
Integrated Development Environments (IDEs), 6, **14–15**
  for debugger setup, **332–336**, *333–335*
  help files for, 602
interdependencies, project manager for, 7
interface builders, Constructor, **19–20**, *20*
interface files, 5. *See also* header files
interface model, human, **116**
Interface pop-up menu
  in Editor window, *113*, 136, *136*, 287, *287*
  in Errors & Warnings window, 312
  in Search Results window, 190
Interfaces directory, 605
intermediate representation (IR)
  by compilers, **21–22**, 539
  optimizing, **518–520**, *519*
interpreters, Java, 539
interrelated files in editor, **136–137**, *136*
intrinsics support, 292
IR Optimizer panel, 289, **518–520**, *519*
is32BitCompatible flag, 572
italics for help text, 38
iterators, C++ library for, 420

# J

Java, **538–539**
  for applets, **543–547**, *544–546*, *548*
  for applications, **548–552**, *549–550*
  classes.zip file for, **448–450**
  debugger with, **331–332**
  framework for, **447–451**, *448–449*, *451*
  importing classes from, **450–451**, *451*
  setting up for, **540–542**, *540–541*
  stationery in, **448**, *448*
  support folder for, 603
*Java API Documentation*, 447
java.applet package, 447
java.awt package, 447
Java compiler, bytecodes from, 22, 538–539
.java extension, 286, 542
Java Generic Library, 448
java.io package, 447
java.lang package, 447
*Java Language Spec*, 447
Java Lib.zip file, 549
java.net package, 447
Java Project panel, **541–542**, *541*
Java Support folder, 603
java.util package, 447
Java Virtual Machine (JVM), **22–23**, 331, 447, **538–539**
jumping to routines, **131–132**, *131*
just-in-time (JIT) compilers, 331, 538–539
justification for text, 500
Jview utility, 549, 551–552

# K

Kernel32.lib file, 94
keyboard
  for insertion point, 114–115
  shortcuts with, 4, 487
keywords
  color for, 126–127
  editing list of, **126–129**, *128*
Kill command, 348, 351
killing programs, **351**

# L

language compilers, 21
  choosing, **282–286**, *283*
  settings for, **290–293**, *291*, *293*
language support, C++ library for, 420
large code model, 562
lasso tool, 494–495
Launchable flag, 283–284
launching
  CodeWarrior, 36
  Constructor, 502–503
  debugger, **336–337**, *338*, 391, *392*

Layout menu commands
Edit Grid, 483
Show/Hide Grid, 481–482
Snap/Don't Snap to Grid, 482
layouts, **476–477**
adding objects to, 505, *505*
containment in, **477–479**, *478–479*
custom panes for, **479–480**, *479*, 504–505
moving panes in, **481–486**, *482–484*
pane properties in, **480–481**, *480*
LBroadcaster class, 452–453
Level option, 568
Lewis, Peter, 543
libraries, **412**
adding, 429–430, *429*
for C, **414–419**, *417*
for C++, **419–423**, *421*
debugging, 350, **384**, 432–435, *433–435*
editing, **426–427**
exercises for, **428–436**
folder for, 604
fundamentals of, **16**, **56–57**, **412–414**
including source code from, 56, 456
for Pascal, **423–426**, *425*
target-dependent, 282
Libraries directory
for Mac OS, 604
for Windows, 605
licenses, 29, 595
Life.bat file, 550
Life game, **543–547**, *544–546*, *548*
Life.html file, 545
Life.java file, 545, 550
LifeApp.cwp file, 549
LifeApp.zip file, 552
LifeApplet.cwp file, 543
Lifetime Analysis option, 520
line endings in editor, **134**, *134*
line numbers for insertion point, 115
line tool, 494, 496
lines (graphics) in browser window, 229, *230*
lines (text)
in regular expressions, **197–198**
selecting, 117
link maps
for Mac OS programming, 564
for Windows programming, 305–306, 521
Link Single Segment option, 564
linked lists, frameworks for, 18
linkers and linking
choosing, **303–304**
with compilers, 281
fundamentals of, 47
for Mac OS programming, **563–565**, *563*, **569–570**, *569*
makefiles for, 7
settings for, **305–306**, *306*
for Windows programming, **520–522**, *521*, 525
lists
of files in groups, 68, *68*
frameworks for, 18
of project files, 60
LListener class, 452–453
local variables
in calling chains, 356
in Program window, 344
in register coloring, **370–371**
viewing values of, **367–369**, *368*
windows for, 374–375
localization, C++ library for, 420
Log window, **346**, *346*, 391
loop invariants, 363
Loop Invariants option, 520
loops, infinite, 351
low-level debugger features, **380–383**, *381–383*
LPicture class, 505
LString class, 453
luminary.h file, 430–431
LumTime.cp file, 525, 527
LView class, 472

# M

Mac-hosted debugger, **578–579**
Mac OS, **556–557**, **565–566**
for applications, **570–578**, *571*, *573–574*, *577*
debugging, 563–564, **578–579**

future of, **557**
moving applications to, **575–576**
PowerPlant for, **451–456**, *454*
PPC Codegen panel for, **567–568**, *567*
PPC Linker panel for, **569–570**, *569*
PPC Project panel for, **566**, *566*
68K CodeGen panel for, **561–563**, *561*
68K Linker panel for, **563–565**, *563*
68K Project panel for, **558–560**, *559*
support directory for, 604
Mac OS Support directory, 604
MacHeaders directory, 604
machine code
in disassembling, **297–299**, *298*
for Windows programming, 518
Machine Code Listing option, 518
macros
for compilers, 299
displaying, 219, 226–227
MacsBug Symbols option, 564
Macutils.hqx file, 578
Main option, 570
Make command, 288, 304–305, 319, 336, 390, 431
Make Transparent command, 499
makefiles, 7, 49
Marker menu commands, *113*, 132, *133*
Add marker, 132, 154, *154*
Remove marker, 133
markers
breakpoints as, 365
in editor, **132–134**, *133*, 154, *154*
marquee tool, 494–495
math.h file, 416
math routines
C library for, 416
Pascal library for, 424
MathLib68K(2i).lib file, 571
MBAR resource, 487
MCWHello file, 574–576
MDEF IDs, 489, 560
members
in class browser, **246**, *247*
in object-oriented code, 217
memory
debugger for, **380–381**, *381*
Pascal library routines for, 424
watchpoints for, **385–386**, *386*
memory attributes for resources, 474
Memory window, 380–381, *381*
Menu Info command, 489
Menu Info dialog box, 489, *489*
Menu Item Info command, 491
menus, 4, **32–33**, *32*, **486–488**, *487*, *491*, 506–508, *507*
for debugger, 339
editing, **490–492**, *491*
exercises for, **36–37**, *37*
making, **488**, 507
menu bars for, **488–490**, *488–489*
submenus, **492**
Merge Compiler Glue Into Segment 1 option, 564–565
Merge Into Output command, 449–450, 547
message windows, background of, **187–188**
methods, 5, 217, 542
Metrowerks C runtime library, 418
Metrowerks Standard Library folder, 604
Metrowerks Web site, 594
MFC 4.2.cwp file, 445
MFC Application stationery, 528
Mfc.lib file, 443–445, 455
(MFC4.2) directory, 605
MFCHeaders file, 444–445
MFCHeaders.pch file, 444–445
Microsoft Foundation Classes (MFC), 17, 440
exercise for, **456–460**
including headers from, **444–445**
precompiling, 300–301
source code in, **445–446**
stationery in, **443–444**, *443*
Minimum Heap Size option, 559
missing breakpoints, **362–363**
modification dates and times, tracking, **70–71**, *71*, 80
modules, 217, 220
Mohr, Brad, 570
monospace fonts, 123
Motorola series processors, 556
mouse with insertion point, 114–115

movable toolbars, 34–35, 37
moving
applications to MAC OS computers, **575–576**
in Constructor, 467
Constructor projects, **475**
files and groups, **66**, *66*
insertion point, **114–116**
layout panes, **481–486**, *482–484*
menu items, 490
menus, 489
text, **118–119**, *118–119*, 147–148, *147*
MPW utility, 560
MSL C.68k(4i).lib file, 417
MSL C library, calling from Pascal, **427–428**
MSL C PPC.lib file, 417
MSL C++.68K.lib file, 422
MSL C++ PPC.lib file, 422
*MSL C Reference*, 415
*MSL C++ Reference*, 420
multi-class browsing, **10–11**, *11*, 219, 222–224, **239–242**, *240*, *242*, 268–270, *268–270*
for class declarations, 256
Classes pane in, **242–244**, *243–244*
Data Members pane in, **246**, *247*
for functions, 257
Functions pane in, **244–246**, *246*
Source pane in, **241–242**, *242*, **247–248**, *248*
for viewing and editing code, **248–249**
multi-class hierarchy view, 228–229, *229*, 265–266, *266*
appearance of, **229**, *230*
for class declarations, 256
deep and wide disclosures in, **233–234**, *234*
for functions, 257
for multiple inheritance, **234–235**, *235–236*
navigating in, **235–238**, *237*
showing and hiding classes in, **230–232**, *231–232*
multiple document interface (MDI) windows, 32
multiple files, searching in, **175–185**, *177–179*, *181*, *183–184*, 205–210, *206–210*
multiple inheritance
browser display for, **234–235**, *235–236*
in PowerPlant, **452–455**, *454*
multiple languages, **20**
compilers for, **21–24**, *21*
debugger for, **387**
multiple projects, debugger for, **388–389**
multiple targets, **587–589**, *588*
multiple undo/redo feature, **120–122**
multiple windows, **137–138**
multiply-defined symbols, **252–254**, *253*
multitasking, debugger for, **387–388**, *388*
Mwcrtl.lib file, 94, 418, 429, 444
MWDebug.exe file, 601, 604
MyMFCapp.cp file, 458
MyMFCapp.h file, 458

# N

name collisions in Pascal, 426
Name new project as dialog box, 53–54, *54*, 89
Name option, 541
names
for files, **140–141**
for groups, **64–65**, *65*, 95, *96*
for Java programs, 541
for Mac OS programs, 541
for markers, 132
for menus, 491
for projects, 53
for resources, 472–473
for source files, 282
spaces in, 285, 551
for text traits, 501
on title bars, 108
for variables, 372
for Windows projects, 515
narrow-scope browser view, 222–223
Native option, 520
navigating code
in browser, **254–257**
in debugger, **355–358**, *356–357*
network support, Java classes for, 447

New command, 109
New Class Browser command, 223, 240, 268
New Menu command, 488, 507
New Menu Item command, 490, 507
New Project command, 50, 88, 96
New Project dialog box, 50–53, *50*, *53*, 88, 96–97, *97*
New Project File command, 469
New Resource command, 469, *470*, 503
New Separator Item command, 490
NeXT, purchase of, 557
null-terminated strings, 428
number signs (#) for preprocessor statements, 299
numerics, C++ library for, 421

# O

object code
    deleting, 100, 288
    generating, 47
object-oriented code
    with browser, 219
    data hiding in, **219–220**
    debugging, 341
    frameworks for, 18, 442
    fundamentals of, **217–218**
Objective C language, 557
objects
    in object-oriented code, 217
    overlapping, **477–479**, *478–479*
    properties of, 505–506, *506*
Open command, 58, 68
Open Array Window command, 377
Open dialog box
    with debugger, 337, *338*
    for projects, 58, *59*
Open File command, 110
Open Selection command, 137
Open Variable Window command, 374
opening
    browser files, 247, *248*
    Find window, 201, *201*
    groups, 65
    project files, **67–68**, *68*, 99, **110–111**, 149–150
    projects, **58–59**, *59*, 100–101
    related files, **136–137**, *136*
operating systems, 5, **280–282**, *281*
Optimize Code Generation option, 541
Optimize For option, 568
Optimize Space option, 518
Optimize Speed option, 518
optimizing
    intermediate representation, **518–520**, *519*
    in Java programming, 541–542
    in Mac OS 68K programming, 562
    in Mac OS PowerPC programming, 568
    missing breakpoints from, **362–363**
    in Windows programming, 518
Options menu commands, *113*
    for line endings, 134
    Make Transparent, 499
    Syntax Coloring, 129, *129*
ordinal operations, Pascal library for, 424
orientation
    in class browser, 242
    for printing, 143
Output File Type option, 515
Output Format option, 541
Output settings for Windows programming, 521–522
overlapping layout objects, **477–479**, *478–479*

# P

.p extension, 286
pad-bytes, 517
Page Down key, 115
Page Up key, 115
paint tool, 494, 498–499
Pane Splitter, *113*
panes
    in editor, **124–125**, *125*, 152–153, *152*
    in layouts, **479–480**, *479*
        aligning, **483–485**, *483–484*

custom, 504–505
distributing, **485–486**
moving, **481–486**, *482–484*
properties for, **480–481**, *480*
in multi-class browser
Classes, **242–244**, *243–244*
Data Members, **246**, *247*
Functions, **244–246**, *246*
Source, **241–242**, *242*, **247–248**, *248*
in Search Results window, 190
in single-class browser, **250–252**, *251*
paper size and source, 143
parallel processing, 585
parameters
in class browser, 249, *250*
for DLLs, 57
parentheses ()
balancing, **129–131**
for folders, 83, 87, 297
in regular expressions, **195–196**
parser
in browser, 216, 218
symbol catalog by, 218–219
.pas extension, 286
Pascal
calling MSL C library from, **427–428**
dependency lists for, 80
libraries for, 16, **423–426**, *425*
link maps for, 306
Pascal Compiler panel, **292–293**, *293*
*Pascal Library Reference*, 423, 425
Pascal String format command, 373
Pascal strings for variable values, 372
Pascal Warnings panel, 309–310, *310*
paths
for applications, **551–552**
in Errors & Warnings window, 312
for projects, **81–84**, *82*, 178, *178*, **293–296**, *294–295*
in Source pane, 247
in symbolics file, 341
pattern tool, 495, 498
patterns in bitmaps, **497–498**
PC Exchange utility, 475, 575
.pch extension, 284
compilers for, 286
including files with, 301–303
.txt with, 320
.pch++ extension, 286
Peephole Optimization option
for Mac OS programming, 568
for Windows programming, 518
pencil tool, 494–495
periods (.) in regular expressions, **194**
pixels
color of, **495–496**
editing, **497**
placeholder files, 85
platforms
converting code between, 134
with debugger, **330–332**
libraries for, 412–413
with PowerPlant, 455
Please Select an Access Path dialog box, 295–296, *296*
*Plugin API Guide*, 280
plugin compilers, **279–280**, *279*
Plugins folder, 279–280, *279*, 601
plus signs (+) in regular expressions, **195**
Pop-up Display Button, *113*
pop-up menus in Project window, 55, 58
Post Linker pop-up menu, 304
post-linkers, 78, 304
PowerPC programming, 556, **565–566**
PPC Codegen panel for, **567–568**, *567*
PPC Linker panel for, **569–570**, *569*
PPC Project panel for, **566**, *566*
*PowerPlant Advanced Topics*, 456, 466
*PowerPlant Book*, 456, 466, 487
PowerPlant directory, 604
PowerPlant framework, 17, 440, **451**, **454–455**, *455*
background of, **452–453**
precompiling, 300–301
PPC Codegen panel, **567–568**, *567*
PPC Linker panel, **569–570**, *569*
PPC Project panel, **566**, *566*
PPob resource, 476, 504–505, *504*

#pragma statements, 134, 299, 523
Precompile command, 301, 321
precompiled files, 56, 284
creating, **300–303**, *301–302*, 319–321, *321*
in MFC, **444–445**
prefix files for, 292–293, 301, 321–323, *322*
Precompiled flag, 282, 284
precompiling resources, 522
Preference dialog box, 38, *39*, 74–75, *74*
for automatic saves, 140
for balancing punctuation, 130
for browser, 260–261, *260–261*
for debugging, 349–351, *349*
for drag-and-drop, 119, *119*, 149
for fonts, 122, *122*, 148, *148*
for multiple undo/redo, 122
for routine sorting order, 132
for syntax coloring, 126–127, *127*
for unbounded arrays, 377, *378*
Preferences command, 38, 74, 119
preferences for projects, **73–75**, *74*
Preferred Heap Size option, 559
prefix files
for precompiled headers, 301, 321–323, *322*
support for, 292–293
Preprocess command, 299
preprocessors for compilers, **299–300**, *300*
Print command, 144
Print Setup command, 143
Print Setup dialog box, 143–144, *143*
printing files, **143–144**
private members option, 251
private methods, 542
procedures, 5
processes, 5
Processes window, 388, *388*
processors
Mac OS programming options for, 562
target compilers for, **280–282**, *281*
Produce.cwp file, 262
profiler calls, setting for, 563
Profiler utility, 578, 604–605
Program Control panel, 349–351, *349*
program counter, current statement arrow as, **353–355**, *354*, *394*
Program window, 337, *338*, **343–345**, *343*, 368, *368*, 391, *392*
progress indicator for compilation, 294, *294*
Project Headers option, 177–178, 206
Project menu commands, 33
Add Files, 64, 94, 321, 417, 429, 457–458
Add Window, 63, 93
Bring Up To Date, 98, 155, 262–263, 288, 315
Check Syntax, 289
Compile, 288, 314
Debug, 288, 304–305, 336–337, 432
Disassemble, 298
Enable Debugger, 332–333, *333*, 336, 390
Make, 288, 304–305, 319, 336, 390, 431
Precompile, 301, 321
Preprocess, 299
Remove Binaries, 100, 288
Remove Files, 67, 458
Run, 288, 304–305, 319, 336
Synchronize Modification Dates, 71
project relative access paths, 296
Project Settings command, 76, 89, 127
Project Settings dialog box
for access paths, 81–84, *82*, 178, *178*, 293–296, *295*
for Build Extras, 79–80, *81*, 224–225, *225*
for C/C++ compilers, 290–292, *291*
for compiler, 11, *12*
for debugger, 334–336, *335*
for Java programming, **540–542**, *540–541*
for keywords, 128–129, *128*
for linkers, 305–306, *306*
for Mac OS 68K programming, **558–565**, *559*, *561*, *563*
for Mac OS PowerPC programming, **565–570**, *566–567*, *569*
for Pascal compilers, 292–293, *293*
for targets, 77–79, *77*, *80*, 89–92, *92*, 280–286, *281*, *283*
for warnings, 308–310, *309–310*
for Windows programming, **515–522**, *516–517*, *519*, *521*
Project Switch List command, 59, *59*, 100–101, *101*
project totals, 55, *55*, 58
Project Type option, 558–559
Project window, **54–58**, *55*

projects and project manager, **46–48**, *47*
access paths for, **81–84**, *82*, 178, *178*, 293–294, *294*
adding files to, **62–64**, *64*, 93–95, *93–95*
adding libraries to, 429–430, *429*
background of, **49**
binary files in, 100
Build Extras settings for, **79–80**, *81*
C library in, **416–419**, *417*
C++ library in, **421–423**, *421*
closing, **60–61**, 100
copying, 60
creating, **49–54**, *50*, *53–54*, 88–89, 96–97, *97–98*
debugging information for, **71–73**, *72*
default options for, **75–76**
elements in, 6, *6*
exercises for, **87–101**
groups for, **64–67**, *65–66*, 95, *96*
included files in, **69–70**, *69*, 99–100, *99*
new version of, **587–589**, *587–588*
opening, **58–59**, *59*, 100–101
opening files in, **67–68**, *68*, 99, **110–111**, 149–150
Pascal library in, **425–426**, *425*
preferences for, **73–75**, *74*
Project window for, **54–58**, *55*
saving, 54, **60**, 89
selecting files in, **61–62**, *62*
settings for, **76**, **79**, *80*
stationery for, **84–87**, *86*, 96, *97*
targets for, **77–79**, *77*, *80*, **89–92**, *92*, 280–286, *281*, *283*, **587–589**, *588*
touching files in, **70–71**, *71*, 287
updating, **98–99**, *98*, 155, 315
properties
for menu items, 490–491, *491*
for objects, 505–506, *506*
for panes, **480–481**, *480*
property inspector window
for menu items, 491, *491*
for panes, 480, *480*
protected option, 251
prototypes
in class browser, 248–249, *250*
as declarations, 227
public option, 251
punctuation, balancing, **129–131**, 150–151, *151*
pure virtual functions, 245

# Q

quotation marks (") for included files, 83, 297

# R

\r character
in regular expressions, 198
in searches, 168
ranges
of memory, watchpoints for, **385–386**, *386*
in regular expressions, **196**
with search and replace, 162, **169**, **180–181**, *181*
of text, selecting, **117–118**
rapid application development (RAD) tool. *See* Constructor
.rc files, 284
recursive file searches, 295–297, *295*
red check marks in editor, 139, *139*, 144, *145*
red circles for breakpoints, 359, 361
Redo command, 120–121, 146
Reduction in Strength option, 520
Regexp option, 168, 192–193
register coloring, **370–371**
registers
A6, 564
viewing and changing, **382**, *382*
Registers window, 382, *382*
registration, 595–596
regular expressions, **191–192**, 210–212, *211*
activating, **192–193**, *192*
ampersands in, **198–199**
asterisks in, **194–195**
backslashes in, **197–198**
brackets and hyphens in, **196**

carets in, **196–197**
dollar signs in, **197–198**
parentheses in, **195–196**
periods in, **194**
plus signs in, **195**
syntax of, **193–194**
working with, **199**
related files in editor, **136–137**, *136*
relative access paths, 296
release notes, folder for, 603
Remove a File Set command, 184
Remove Binaries command, 100, 288
Remove Files command, 67, 458
Remove marker command, 133
Remove Markers dialog box, 133, *133*
removing. *See* deleting
renaming
files, **140–141**
groups, 65
repeating code, **353–355**, *354*
repetitions in regular expressions, **194–196**
Replace command, 173, 190
Replace All command, 173–174, 181
Replace & Find Next command, 173
Replace & Find Previous command, 173
replacement text in search and replace, 161–162, **166–168**, *167*
replacing elements. *See* search and replace feature
.res extension, 284, 522
ResEdit editor, 576
Reserved Heap Size option, 516
Reserved Stack Size option, 516
Reset Toolbar command, 39
resetting toolbar, 35, 39
resize bars, 312
resource compiler, **522**
Resource File flag, 283–284
Resource.frk folder, 91, **474–476**, 575–576
Resource Info command, 474
resources
in Constructor, 466–467
directory for, 604
making, **469–472**, *470–471*
precompiling, 522
and resource forks, 91, **474–476**
working with, **472–474**, *474*
Resources directory, 604
return characters
in regular expressions, 198
in searches, 168
reusing code. *See* libraries
Revert command, 141
reverting files, **141**
right-angle lines in browser window, 229, *230*
RISC (reduced instruction set chips), 565
Routine pop-up menu
in Editor window, *113*, 131, *131*, 153, *153*
in Search Results window, 190
in Source Browser window, 342
routines, 4–5
in browser, 218, 226–227, **244–246**, *246*, **256–257**
calling chains for, **344**
defined, 5
jumping to, **131–132**, *131*
libraries for. *See* libraries
stepping into, **352–353**, 394–395, *396*
stepping out of, **353**
stepping over, **352**, 393
Routines.c file, 149, *149*, 152, *152*, 155–156, 199–201
Routines.h file, 155–156
.rsr extension, 469, 475
.rsrc extension, 573
Run command
for debugger, 343, 348, 350
for projects, 288, 304–305, 319, 336
running
applets, **547**, *548*
applications, **530–531**, **552**, **577**, *577*
debugger code, **348–350**, *349*


# S


sales department, **595**
sample bitmap views, **496–497**

Save command, 139, 145
Save a Copy As command
for Errors & Warnings window, 313
for hit lists, 191
for projects files, 60, 141
Save All command, 139
Save All Before "Update" option, 140
Save As command, 140–141
Save Default Window command, 136, 142, 150, 269
Save File Set As dialog box, 182, *183*
Save Settings command, 378
Save Settings in Local ".dbg" Files option, 379
Save this File Set command, 182, 206
saving
assembly-language code, 298
automatic, **139–140**
changes, 109
debug information, **378–379**
Errors & Warnings window, 313
file sets, 182, 206–207, *207*
files in editor, **138–142**, *139*, 145–146
projects, 54, **60**, 89
window settings, **141–142**
scaling bitmaps, 497
scope with search and replace, 162, **170–171**
scroll bars in panes, 125
search and replace feature, **160–161**
adding files for, **179**
batch reports for, **185–191**, *186*, *189*, 207–210, *208–210*
controls for, **162–164**, *163*
deleting files from searches, **180**
design of, **161–162**
exercises for, **199–212**
file sets for, **182–185**, *183–184*, 206–207, *207*
without Find window, **174–175**
finding and replacing text in, **171–174**, 200–203, *200*, *202–203*
for multiple files, **175–185**, 205–210, *206–210*
ranges with, 162, **169**, **180–181**, *181*
regular expressions for, **191–199**, *192*, 210–212, *211*
scope with, 162, **170–171**
for selected text, **164–166**
selecting files for, **176–178**, 205–206, *206–207*
for single files, **166–175**, *167*
special characters in, **168**
Search menu commands, 33, 162, *163*
Enter 'Find' String, 166
Enter 'Replace" String, 167, 173
Find, 162, 171, 186–187, 201
Find in Next File, 182
Find Next, 165, 171, 200
Find Previous, 165, 171, 201
Find Previous Selection, 165
Find Selection, 164–165, 200
Go Back, 258, 273
Go Forward, 258, 273
Replace, 173, 190
Replace All, 173–174, 181
Replace & Find Next, 173
Replace & Find Previous, 173
Search Results window, **187–190**, *189*, 208–210, *209–210*
background of, **187–188**
editing in, **190–191**
search strings, **166–168**, *167*
searches
in code, **357–358**, *357*
in compilation, 295–297, *295*
in editor. *See* search and replace feature
Select files to add dialog box, 64, *64*, 94, 429, *429*
Select files to search dialog box, 179, *179*
Select Type dialog box, 381, *381*
selected text, finding, **164–166**
selecting
vs. choosing, 4
in Constructor, 467
files in projects, **61–62**, *62*
text, **117–118**, 145
semicolons (;)
compiler warnings for, 309
errors involving, 316–317
Sentence.cwp file, 390–392, *392*
separator bars for menus, 490
Set Watchpoint command, 385
settings, 73

Settings & Display panel, 341
for data types, 372
for saving debug information, 379
for unbounded arrays, 377, *378*
Settings dialog box. *See* Project Settings dialog box
shadows for resource forks, **91**, 474
shared libraries, 560
Shift-clicking, 117
Shift key
with drawing tools, 496
in selecting files, 61
in selecting text, 117–118
Shift Left command, 123, 152
Shift Right command, 123, 152
shifting indentation, **123–124**, 151–152, *151*
shortcuts, installation of, 29–30
Show Breakpoints command, 364, 400
Show Catalog Window command, 223, 225
Show Declaration option, 271
Show Expressions command, 375, 403
Show FPU Registers command, 382
Show/Hide Grid command, 481–482
Show Hierarchy option, 271
Show Hierarchy Window command, 223, 228
Show Processes command, 387–388
Show Registers command, 382
Show Toolbar command, 37
Show Types command, 372, 402
Show Watchpoints command, 386
sibling classes, displaying, 229, 233
Signed Decimal format command, 373
single-class browsing, **249–252**, *251*, 270–272, *271–272*
for class declarations, 256
for functions, 257
single-class hierarchy view, 236, *237*, **238–239**, *239*, 266–268, *267*
for class declarations, 256
for functions, 257
single files, searching in, **166–175**, *167*
site licenses, 595
68K CodeGen panel, **561–563**, *561*
68K Linker panel, **563–565**, *563*
68K Project panel, **558–560**, *559*
size
of fonts, **122–123**
of layout grids, 483, *483*
of panes, 125, 241
of paper, 143
of windows, saving settings for, 141–142
'SIZE' flags option, 559, 572
skipping code, **353–355**, *354*
SlideShow.rsr file, 503
small code model, 562
smart code model, 562
Snap/Don't Snap to Grid command, 482
Solution Code folder, 87
Sort Function Popup option, 132
Sort Functions by Method Names in Browser option, 341
sorting
classes in Class pane, 243, *243*
routine names, 131–132
Source Browser window
for code, **356–357**
in debugger, 337, *338*, **340–343**, *340*
for variable values, 367–368, *368*
source files
for applets, **544–545**, *545*
for applications, **529–530**, *530*, **550**, *550*, **572–574**, *573*
defined, 5
for DLLs, **525–526**, *526*
editor for. *See* editor
including, 456
for Java programming, 541
for libraries, 418, **426–427**
in MFC, **445–446**
for multiple targets, 589
names for, 282
searching in, **177–179**, 206
switching with debugger, **358–359**
source-level debuggers, 328
Source Model pop-up menu, 291
Source pane
in class browser, **241–242**, *242*, **247–248**, *248*
in Errors & Warnings window, 311–312, *311*
in Source Browser window, 342

Sources search option, 177, 206
spaces
in directory names, 551
in file names, 285
special characters
in regular expressions, **198**
in searches, **168**
speeding up compiling, **293–297**, *294–296*
splitting
groups, 65
panes, **124–125**, *125*, 152–153, *152*
Spread pane distribution options, 486
square brackets ([])
balancing, **129–131**
in regular expressions, **196**
squares, drawing, 496
Stack command, 137–138
stack crawl pane
examining, 350
fundamentals of, **344**
walking through, **355–356**, *356*, 395, *396*
stack frames, 564
Stack pane, 395, *396*
stack pointer, 564
stacks
and calling chains, 344
size of, 516
standard libraries. *See* libraries
standard object model (SOM), 291
Standard Template Library (STL), 419, 421, 423
Start Code folder, 87
Startup Code option, 559
static constructors, debugging, 349
static data members, 246
static functions, 245
static-link libraries, **56–57**
static methods, 542
stationery, **50–53**, *53*
background of, **52**
customizing, **84–87**, *86*, 96, *97*
directory for, 603
in Java, **448**, *448*
in MFC, **443–444**, *443*
for targets, 51–52, 281
(`Stationery`) folder, 86–87, *86*, 96, *97*, 603
stdio.h file, 415
stdlib.h file, 415
Step Into command, 348, 352–353, 394
Step Out command, 348, 353
Step Over command, 348, 352, 393
Stepanov, Alexander, 419
stepping
through code, **347–355**, *347*, *353–354*
into routines, **352–353**, 394–395, *396*
out of routines, **353**
over routines, **352**, 393
*STL Tutorial and Reference Guide*, 421
Stop at Beginning of Main When Launching Applications option, 350
Stop at End of File option, 182
Stop command, 348, 350–351
stopping programs, **350–351**
Store Analysis Results option, 80
Stress.exe utility, 601
string.h file, 416
strings
C library for, 416
C++ library for, 420
C-style vs. Pascal-style, **428**
color for, 126
Pascal library for, 424
for variable values, 372
Struct Alignment option
for Mac OS 68K programming, 562
for Mac OS PowerPC programming, 567
structures
aligning, 562, 567
frameworks for, 18
viewing values of, **368–369**
student pricing, 595
styles for text, 500
subclasses in browser window, 228–229
Submenu ID field, 492
submenus, **492**

subroutines. *See* routines
SubSystem IDs, 521
SubSystem pop-up menu, 92, 520–521
support, **594–595**
Suppress Warning Messages option, 564
switching
  between debugger and source code, **358–359**
  panes, 125
  projects, 59, *59*, 100–101
.SYM extension, 329, 336
SYM files, 305–306, 329, 390–391
  for Mac OS programming, 564
  for Windows programming, 518
SYM Format option, 518
Symbol window
  in browser, **252–254**, *253*, 272–273, *273*
  for functions, 257
symbolics file
  for debugger, 328–329
  paths in, 341
symbols
  in browser, 218, 226–227, **252–254**, *253*
  context menu for, 238
Synchronize Modification Dates command, 71
syntax coloring, **126–129**, *127–129*, 148
Syntax Coloring command, 129, *129*
Syntax Coloring panel, 126–127, *127*
system header files, searching in, **177–179**
System Headers option, 177–178
System Include Path option, 81–82
System qualifier, 426
system relative access paths, 296

# T

\t character
  in regular expressions, 198
  in searches, 168
tab characters
  in regular expressions, 198
  in searches, 168
  setting number of spaces for, 122
Tab key
  in class browser, 241
  in Constructor, 468
  in Errors & Warnings window, 316, *317*, 318
  in Search Results window, 191
tables, frameworks for, 18
target compilers, **280–282**, *281*
Target panel, 11, *12*, 77–79, *77*, 89–92, 280–286, *281*, *283*, **540**, *540*
target text in search and replace, 161–162
*Targeting Java*, 447
*Targeting Mac OS*, 560, 562, 567
*Targeting Win32*, 292, 523, 527
targets, **20–24**, *21*
  defined, 5
  multiple, **587–589**, *588*
  for projects, **77–79**, *77*, *80*, **89–92**, *92*, 280–286, *281*, *283*
  stationery for, **51–52**
technical support, **594–595**
template classes in C++ library, 419
templates
  displaying with browser, 219, 226–227
  stationery for. *See* stationery
temporary breakpoints, **360–361**, 394, *395*
Termination option, 570
termination routines, Pascal library for, 424
tests in conditional breakpoints, 366
text
  adding, **113–114**, 144–145
  color for, **126–129**, *127–129*, 148, 150, 500
  in Constructor, 468, **500–501**, *501*, 503–504, *504*
  deleting, **120**
  moving, **118–119**, *118–119*, 147–148, *147*
  for search and replace strings, **166–168**, *167*
  searching for. *See* search and replace feature
  selecting, **117–118**, 145
text tool, 494
text traits, **500–501**, *501*, 503–504, *504*
third-party utilities, 601–602
Tile command, 137–138
Tile Vertical command, 137–138, 155–156

tiling windows, **137–138**, 155–156
time functions, C library for, 416
time.h file, 416
Timestep.cp file, 525, 527
title bars, 108
titles of menus, 489
toolbars, 4, **33–35**, *34*
  clearing, 39
  exercises for, **37–40**, *38–39*
  resetting, 35, 39
Tools directory, 604–605
Tools menu commands, 33–34, *35*
  Clear Toolbar, 39
  Hide Toolbar, 38
  Reset Toolbar, 39
  Show Toolbar, 37
  Unanchor Toolbar, 35, 37
Touch column, *55*, 70–71, *71*
Touch command, 69, 71, 287
touching files, **70–71**, *71*
tracking files. *See* projects and project manager
trailing spaces in file names, 285
transparency in bitmaps, **499–500**, *499*
Treat All Warnings as Errors option, 314
Treat #include <...> as #include "..." option, 83, 297
triangles
  disclosure. *See* disclosure triangles
  for multiple inheritance, 234
  in PowerPlant, 454
TrivialApplet.html file, 543
TrivialApplet.java file, 543
TrivialApplication.bat file, 549
TrivialApplication.java file, 549
Turbo Pascal language extensions, 523
.txt extension with .pch, 320
type-ahead selection, 62
Type option
  for Java programming, 541
  for Mac OS programming, 559
typedefs in browser, 219, 226–227

# U

unacceptable characters in regular expressions, **196–197**
Unanchor Toolbar command, 35, 37
unbounded arrays, 377, *378*
Undo command, 120–121, 146
undoing changes, **120–122**, 146
Unsigned Decimal format command, 373
untitled windows, 109
Untouch command, 69, 71, 287
Unused Arguments warnings, 308
Unused Variables warnings, 308
updating
  CodeWarrior, 595–596
  DLLs, 57
  projects, **98–99**, *98*, 155, 315
Use Modification Date Caching option, 80
Use Multiple Undo option, 122
User Include Path option, 81–83, 297
User Specified option, 520
User32.lib file, 94
UserID, 521
USES clause, 427
Utility.cp file, 525

# V

variables
  array, **376–377**, *376–377*, 402–403, *403*
  in breakpoint conditions, 366, 401, *401*
  in browser, 218, 226–227
  in calling chains, 356
  data types for, **372–374**, *373*, 401–403, *402–403*
  in debugger, 357, **367–368**, *368*, 397–398, *398*
  Expression window for, **375–376**, *375*, 403–404, *404*
  in object-oriented code, 217
  in optimizing, 363
  in Program window, 344, 368, *368*, 394–395, 397
  in register coloring, **370–371**

setting values of, **369–372**
in Source Browser window, 357
watchpoints for, **384–387**, *386–387*
windows for, **374–375**, *374*
Variables pane, 368, *368*, 394–395, 397
vertical orientation in class browser, 242
video capture utility, 601
View Memory command, 380, 385
View Memory As command, 381
View Type pop-up menu, 470, *471*
viewing toolbar, 35
views
for bitmaps, **496–497**
in browser, **222–224**, *223*, **254–255**
in Constructor, 465
virtual functions, 245
virtual machines, **22–23**, **538–539**
Visual C++, CodeView format with, 337
visual interface in browser, 216, 219


# W


walking
through calling chains, **355–356**, *356*, 395, *396*
through code, **355**, *356–357*
warnings, 12, *13*, **307–308**, **313–314**
Errors & Warnings window for, **310–312**, *311*, 315–319, *316–318*
in Mac OS programming, 564
settings for, **308–310**, *309–310*
watchpoints, **384–385**
setting and clearing, **385–386**, *386*
viewing, **386–387**, *387*
Watchpoints window, 386–387, *387*
Web sites
code for, 600
Metrowerks, 594
whole words
with regular expressions, 193
in searches, 170–171, 202, 358
wide disclosures in browser, **233–234**, *234*
wide-scope view in browser, 222–224
Win32-x86 Support folder, **605**
Win32ResHeaders++ file, 522
Win32x86 target, 289–290, 305
Window menu commands, 33
New Class Browser, 223, 240, 268
Project Switch List, 59, *59*, 100–101, *101*
Save Default Window, 136, 142, 150, 269
Show Breakpoints, 364, 400
Show Catalog Window, 223, 225, 264
Show Expressions, 375, 403
Show FPU Registers, 382
Show Hierarchy Window, 223, 228
Show Processes, 387–388
Show Registers, 382
Show Watchpoints, 386
Stack, 137–138
Tile, 137–138
Tile Vertical, 137–138, 155–156
Zoom, 137–138
windows
in Constructor, 465
cycling through, 138, 155
for files, **108–109**
moving text between, 148
multiple, **137–138**
saving settings for, **141–142**
splitting, **124–125**, *125*, 152–153, *152*
tiling, **137–138**, 155–156
Windows CUI option, 520
Windows GUI option, 520
Windows menu command, Alignment Palette, 483
Windows programming, **514**
for applications, **528–531**, *529–530*
debugging, **531–534**, *532–533*, *535*
for DLLs, **523–528**, *524*, *526–527*
features in, 522–523
IR Optimizer panel for, **518–520**, *519*
setting up for, **514–515**
WinRC Compiler panel for, **522**
x86 CodeGen panel for, **517–518**, *517*

x86 Linker panel for, **520–522**, *521*, 525
x86 Project panel for, **515–516**, *516*
Windows SDK, precompiling, 300–301
WinRC Compiler panel, 289, **522**
WinRes Import tool, 522
word wrap in editor, 114
words
with regular expressions, 193
in searches, 170–171, 202, *203*, 358
selecting, 117
Wrap option, 169, 181, 203
wrapping
in editor, 114
in searches, 169, 173–174, 181, 203–204, *204*, 358
`www.metrowerks.com` site, 594

# X

x86 CodeGen panel, 289, 334, *335*, 390, **517–518**, *517*
x86 Linker panel, 92, *92*, 289, 305–306, *306*, 334, *335*, 390, **520–522**, *521*, 525
x86 Project panel, 79, *80*, 90, 289, **515–516**, *516*

# Z

zip files for Java output, 542, **546–547**
ZoneRanger utility, 578, 604–605
Zoom command, 137–138